SUMC FAULT TOLERANT COMPUTER SYSTEM

FINAL REPORT

FOR

CONTRACT NAS8-31747

September 10, 1980

IBM

SUMC Fault Tolerant Computer System
Final Report
for
Contract NAS8-31747

September 10, 1980

Prepared For:

The Marshall Space Flight Center
Huntsville, Alabama

CONTENTS

ILLUSTRATIONS

1.0 INTRODUCTION

This report presents the results of the trade studies conducted on contract number NAS 8-31747. These trades cover: establishing the basic configuration, establishing the CPU/memory configuration, establishing an approach to crosstrapping interfaces, defining the requirements of the redundancy management unit (RMU), establishing a spare plane switching strategy for the fault-tolerant memory (FTM), and identifying the most cost effective way of extending the memory addressing capability beyond the 64 K-bytes (K=1024) of SUMC-II B. The results of the design are compiled in Contract End Item (CEI) Specification for the NASA Standard Spacecraft Computer II (NSSC-II), IBM 7934507. The report also presents, in Section 3, the implementation of the FTM and memory address expansion. The scope of the original contract was reduced so that the IOUs and RMU were not designed.

2.0 TRADE STUDIES

2.1 CONFIGURATIONS

The first item requiring resolution in the SUMC-II C is the basic configuration. The objectives and constraints of the program are listed below:

- Provide flexibility to be able to meet varying levels of redundancy.
- Make maximum use of hardware developed for the HTC/SUMC-II B.
- Use error correcting memory techniques since they provide the most reliability for the least amount of hardware duplication.
- Use spare memory planes and a bit-plane organized memory.
- Use a fault-tolerant redundancy management unit (RMU) to control the redundant elements of the system.

Providing redundancy within the CPU or IOU would require development of completely new untis and would not represent a cost effective way of providing reliable systems. Configurations were limited to CPUs and IOUs as "stand alone" units. The CPU to memory interface is a significant part of the study. The Top Level system configuration is shown in Figure 2-1. Subsequent

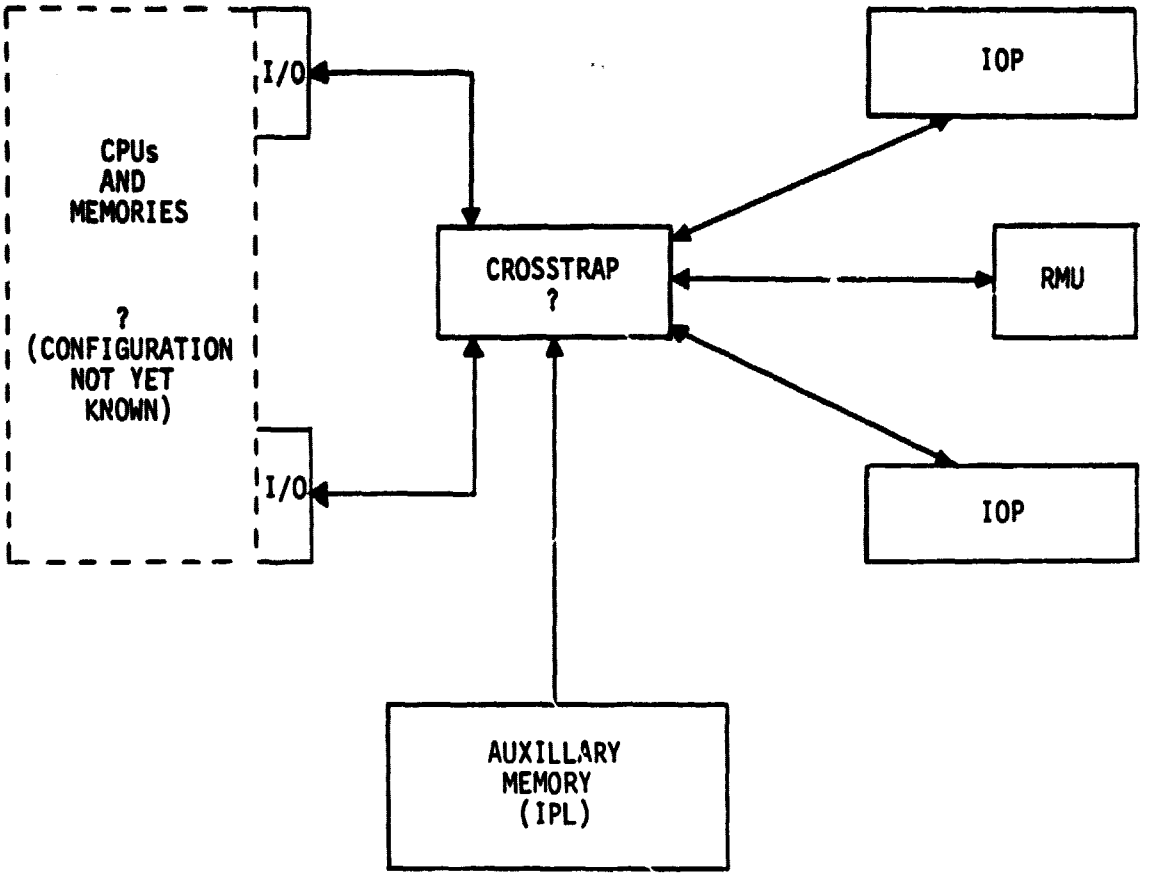


FIGURE 2-1. SIMPLIFIED SUMC-IIC

sections of this report will develop this general concept into a working approach to the SUMC-II C.

Over a period of time a number of candidate subsystems were defined for the CPU/memory portion of the SUMC-II C. These configurations are shown in Figure 2-2 and are explained below:

- Configuration 1 consists of two CPUs sharing a common fault tolerant memory built from existing Main Memory Unit (MMU) hardware. No attempt is made in this configuration to eliminate single point failures in the memory. A new design is required for the power supply and the dual port interfaces.

- Configuration 2 consists of two CPUs sharing a common fault tolerant memory. This memory is an extensively redesigned MMU and contains no single point failures. A new redundant power supply would be required for this design. A new design would be required for the dual port interfaces.

- Configuration 3 consists of two CPUs, each having a fault tolerant MMU memory. The computers will be used in a redundant manner and no new design is required.

- Configuration 4 consists of two CPUs, each having a dedicated memory except that the dual port interfaces are used to provide cross-strapping of memory write when both systems are powered on. Each computer supplies power to its dedicated memory and reads only from its memory. The cross-strapping is provided for rapid

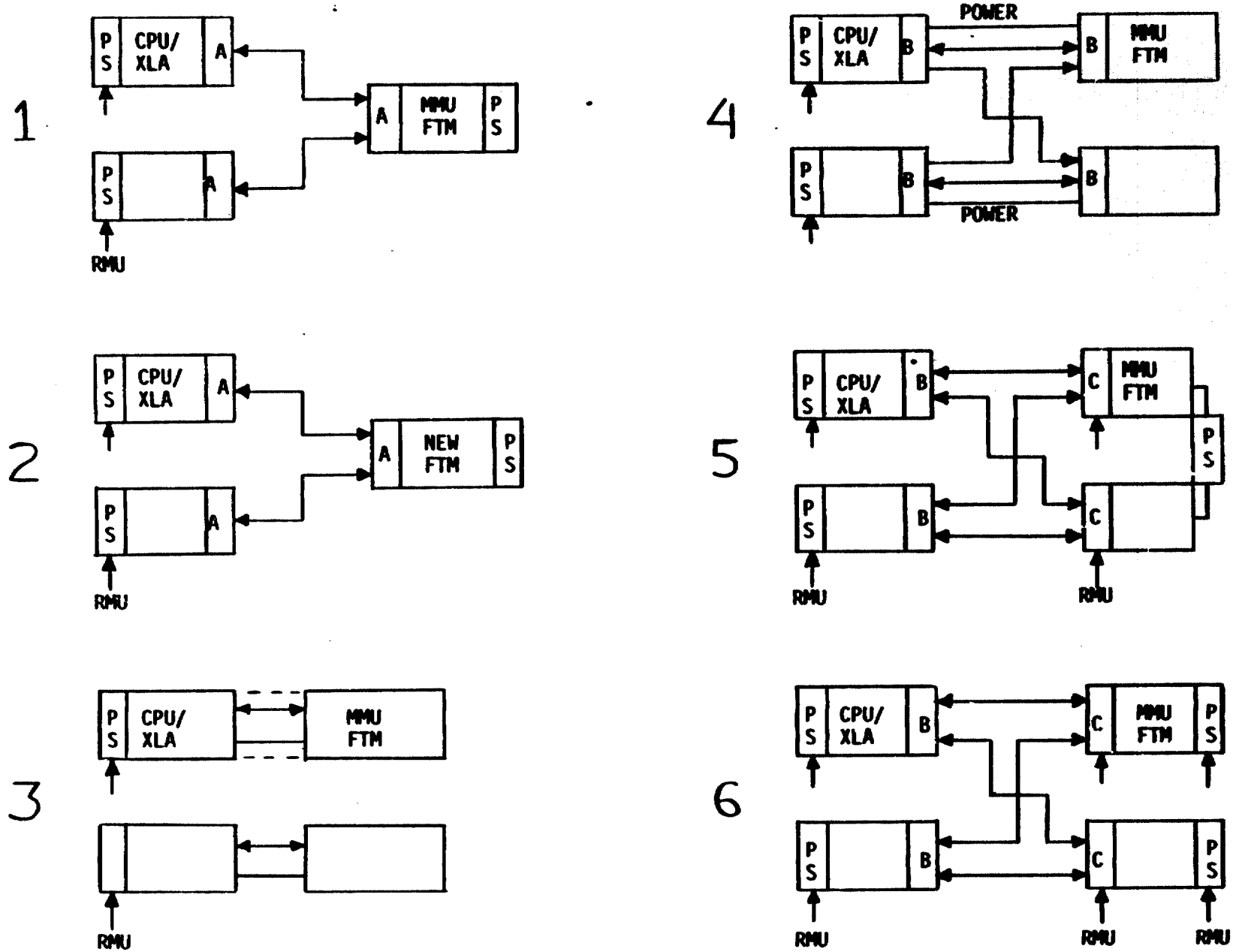


FIGURE 2-2. TRADE-OFF CONFIGURATIONS

information transfer between memories during CPU switch-over. This approach requires a new design for the dual port interfaces but no new power supply design.

- Configuration 5 consists of two CPUs cross-strapped to two fault tolerant MMU memories. Both memories share a redundant power supply and remain in a powered on state. Either CPU can read from either memory but always writes both. The dual port design is used for cross-strapping and the memory read selection is delegated to the Redundancy Management Unit (RMU). A redundant power supply and interface designs are required for this configuration.

- Configuration 6 is the same as configuration 5 except that each memory has its own power supply which can be powered up or down by the RMU to save power. A new power supply and new interface designs are required.

All configurations use common designed interface boards which can be populated/depopulated and/or jumpered to fit each requirement. Figures 2-3 through 2-5 show the interfaces except for the clock switching circuitry. Two CPU interfaces (CPA, CPB) and three memory interfaces (MPA, MPB, MPC) are shown.

The trade data derived for the six configurations is summarized in Table 2-1. Based on that data, configuration three ws selected. The slight increase in reliability of six over that of three does not justify the major difference in development cost. The resultant configuration is shown in Figure 2-6.

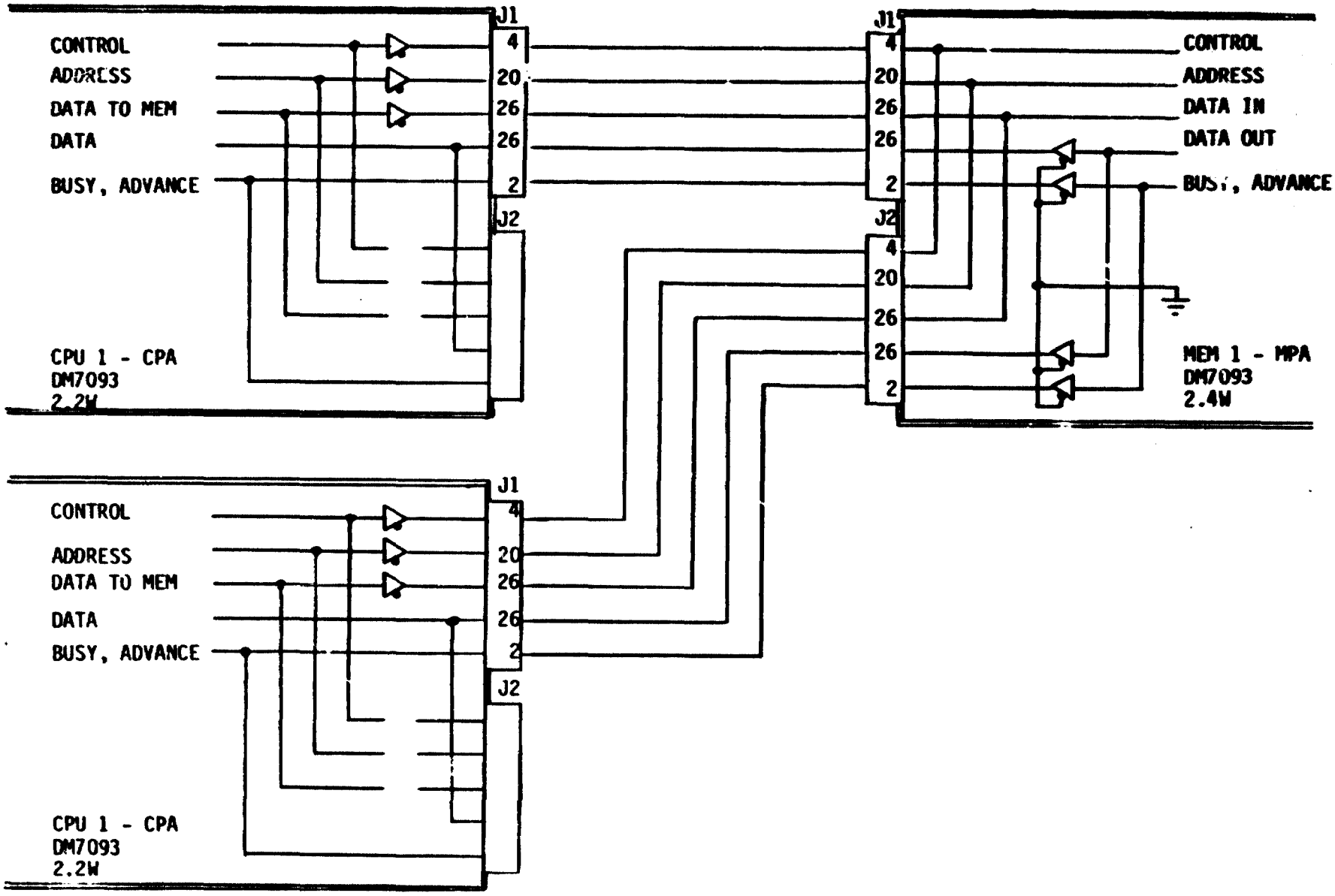


Figure 2-3. Dual Port Configurations 1 and 2

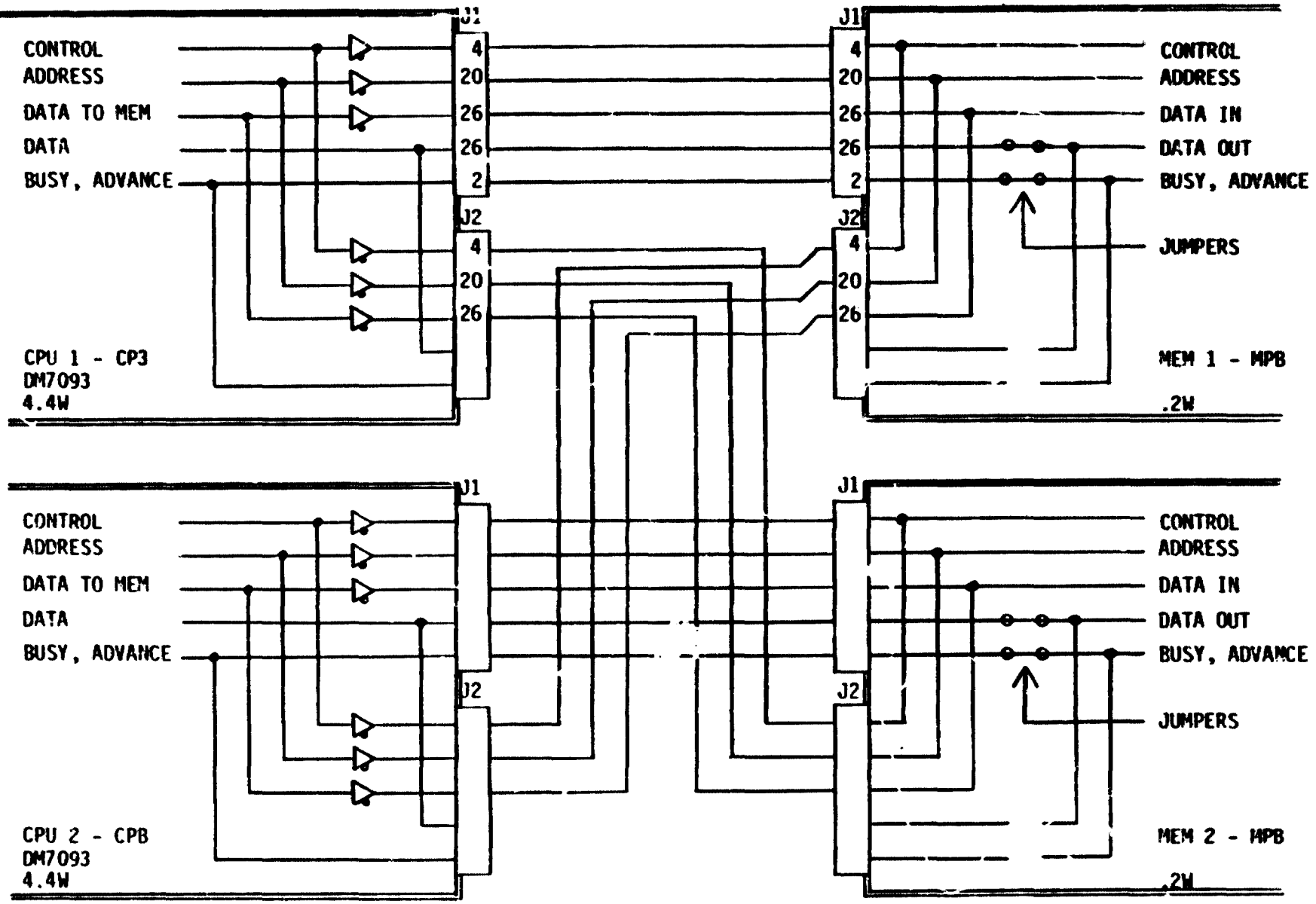


Figure 2-4. Dual Port Configuration 4

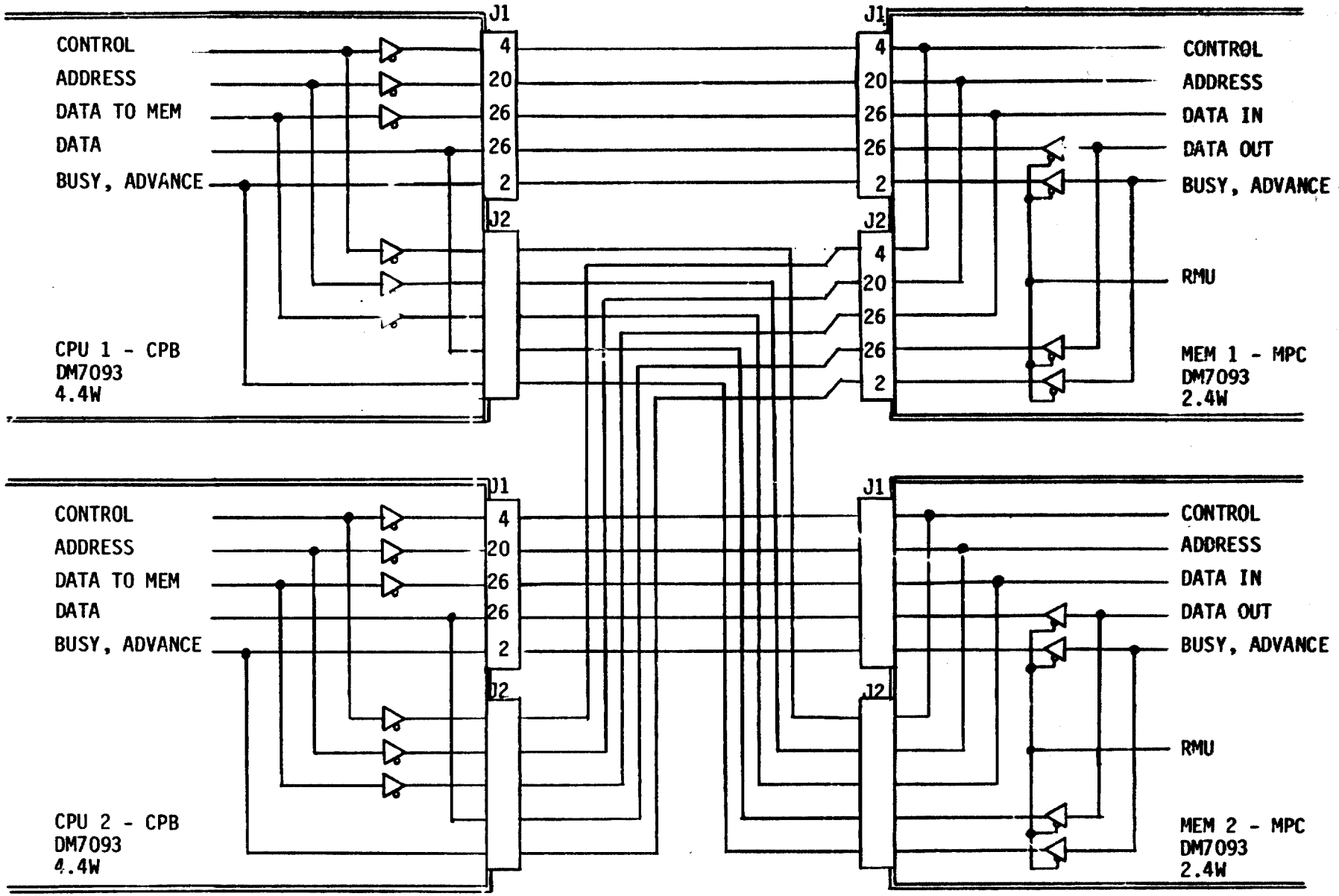


Figure 2-5. Dual Port Configurations 5 and 6

Table 2-1. Configuration Trade Data

CONFIGURATION NO.	1	2	3	4	5	6
New MIB Designs*	2	2	0	2	2	2
New Frame Designs*	2	2	0	2	2	2
No. of Boxes	3	3	2	4	5	4
Typical Power*	131/209	151/229	126/252	131/261	185/266	133/185/266
Reliability (1 yr.)	0.867	0.969	0.971	0.969	0.972	0.982
Single Point Failures*	22FP	No	No	No	No	No
Power Supply Redesign	Yes	Yes	No	No	Yes	Yes
RMU Control of MEM Power	No	No	No	No	No	Yes
RMU Control of MEM Reads	No	No	No	No	Yes	Yes
Relative Cost CPU/MEM	2	10	0	2	2	2
P S	8	10	0	2	10	8
Advantages	Updated Memory	Updated Memory	Existing Design	Switchover Update	Updated Memory	Switchover Update
		High Reliability	Redundant Configuration	Redundant Configuration	High Reliability	High Reliability
			High Reliability		Redundant Memories for each CPU	Redundant Memories for Each CPU
Disadvantages	Lowest Reliability	High Dev. Cost on MEM.	Switchover Updating	Slightly Lower Reliability than 3	High Power	Additional RMU Controls
	New Power Supply	New Power Supply			New Power Supply	New Power Supply
	Dual Port Design	Dual Port Design		Dual Port Design	Dual Port Design	Dual Port Design

*NOTE: Does not include Power Supply

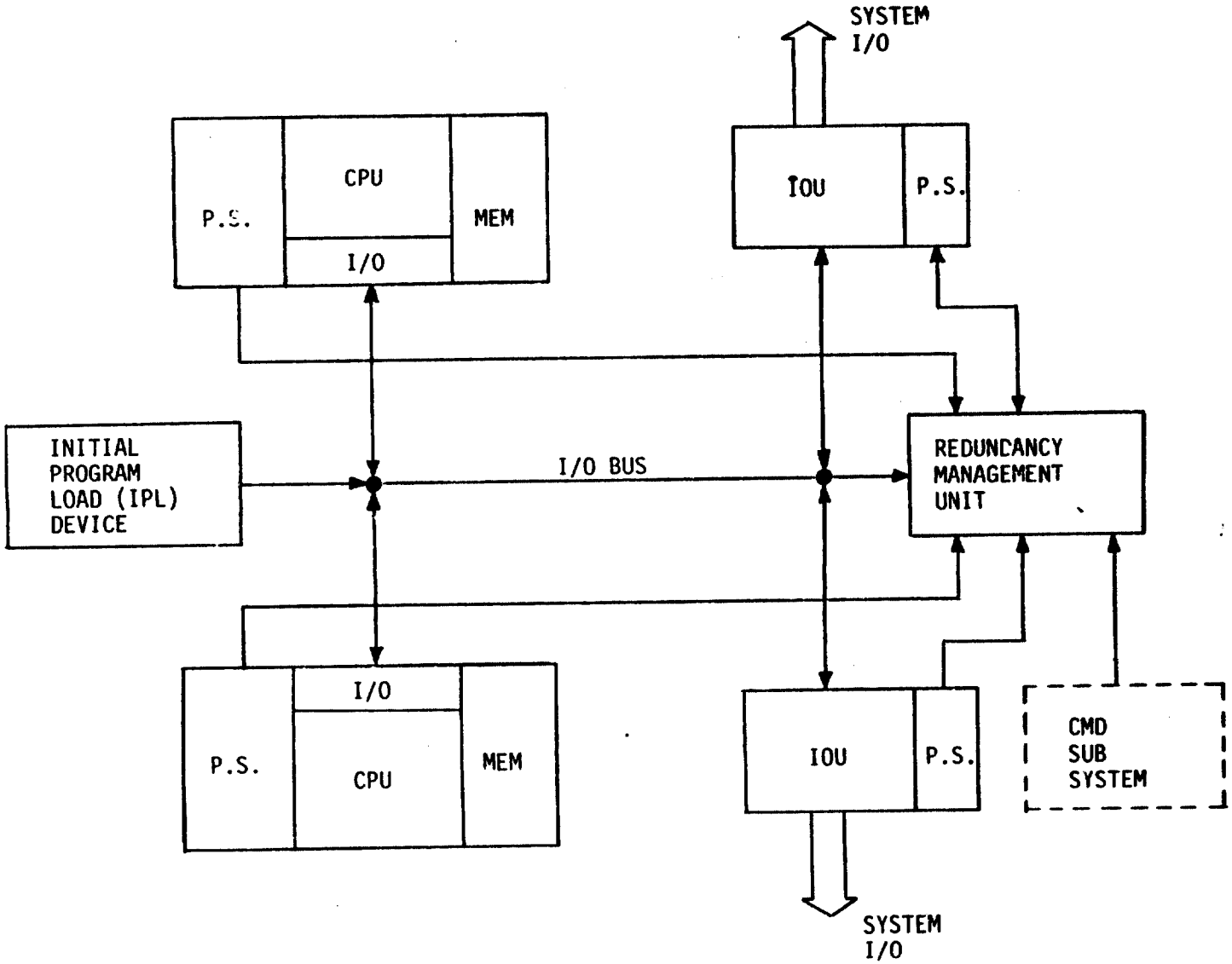


FIGURE 2-6. SUMC-IIC CONFIGURATION

2.2 REDUNDANCY MANAGEMENT UNIT (RMU) CONCEPT

Optimization of the RMU must be done at the system level and is, therefore, application dependent. Several major items which are application dependent are: the time allowed for system recovery, the need for a degraded mode of operation, the availability of program load capability, the degree of operator participation vs. automatic operation, the need for complex interface safing, the need for selecting from among multiple prime power sources, and the need to store recovery parameters for transferring control from one computer to another.

Since mission success is dependent upon the operation of the RMU, it must be essentially failure proof. If the RMU performs many functions, then its simplex implementation requires significant amounts of hardware and it is both difficult and costly to achieve the required reliability.

Effective management of system redundancy can involve many activities. A list follows which represents candidate features which might be included in an RMU:

- Comparators for checking the outputs of multiple computers. This could be required for instantaneous recovery or where the consequence of an undetected error is catastrophic.

- Storage of present status parameters for assistance in restarting after a detected failure.

- Safing all critical spacecraft functions which are under control of the computer.

- Check "status" signals from the computer to assure that all critical program segments were executed and in the correct sequence.

- Gather status and send it to operator/monitor personnel in the spacecraft or on the ground.

- Provide auxiliary storage for error and status logging, program reload, and diagnostic program storage.

- Provide command facilities for remote (probably ground) personnel to override the automatic redundancy management features.

- Provide an independent "Watchdog" timer to detect complete loss of computer operations or excessive time in completing a phase of operations.

- Provide highly redundant (fail-safe) crosstrapping between/among redundant system elements.

- Control application of power to each redundant element of the system.

- Enable/Disable computer I/O.

Since high level redundancy is required in the RMU to meet the necessary failure tolerance, even small amounts of functional

hardware can have significant impacts on power, weight, etc. Therefore, two precepts are proposed for the design of the RMU:

1) Don't include any function which can be performed outside the RMU.

2) Make provisions for but don't include functions which are not required by all applications.

Applying these rules to the candidate list gives the following results:

- o Accommodation should be made for the future addition of compartors but don't include them.

- o If required for rapid recovery on a specific mission, a few restart parameters might be included. A large number should be put in a system storage device and not included in the RMU. The basic RMU should not include restart storage.

- o By putting all I/O, into a predetermined state when system confidence has been lost, proper system design can insure the spacecraft to be safe at all times.

- o Status checking and proper program sequencing are important, however, they should be handled by software checking not the RMU.

- o General status collection should be done by the operating computer which can send it to the ground via telemetry if desired.

- Auxiliary storage should be provided as a system function not by the RMU.

- The ability to use human intervention to override the RMU is desirable and should be provided both for test purposes and as a "last ditch" precaution.

- A "watchdog timer" is an essential part of any RMU.

- Crosstrapping should only be provided in the RMU when required by the application and there is no other place for it.

- The RMU should control power to each unit, although the power switches themselves need not be a part of the RMU.

- The RMU should have the ability to enable/disable outputs. Between the time that power is applied to a computer and the time the computer has initialized and performed self-test, the RMU should insure that all I/O is disabled.

This results in a basic RMU which can be used "as-is" or can be expanded to suit special requirements. The basic RMU features are summarized below.

BASIC RMU SUMMARY

- Watchdog Timer

- Ground Commanded Override of RMU

- Power Control to all Units

- o I/O Enable/Disable Capability
- o Growth for Special Missions
- o Fail Safe Implementation

To communicate with the RMU, the SUMC-II C CPU will use a dedicated line FLAG RMU which will indicate that the output bus has data for the RMU. FLAG RMU is just a pulse so the RMU must store the data on the bus or react quickly to it. In addtion to an "IM OK" code, several codes are generated to signal operational status of the microprogram handling the FTM analysis and spare plane switching. This is explained more fully in Section 3.1.2.

2.3 INTERFACES

Crosstrapping and other redundancy considerations can impact the design of unit-to-unit interfaces. If mission reliability or application design groundrules will not tolerate single point failures, the interfaces used must be carefully designed to prevent a short in one circuit or wire from dragging down an entire function and precluding operation of that functon. The interface shown in Figure 2-7 illustrates a design approach to eliminate all single point failures in a crosstrapped interface.

The circuit of Figure 2-8 is much simpler than that of 2-7 since all drivers and receivers are the ones which would normally be provided in a simplex unit if one precaution is taken in selection of the three state driver. The driver must be selected such

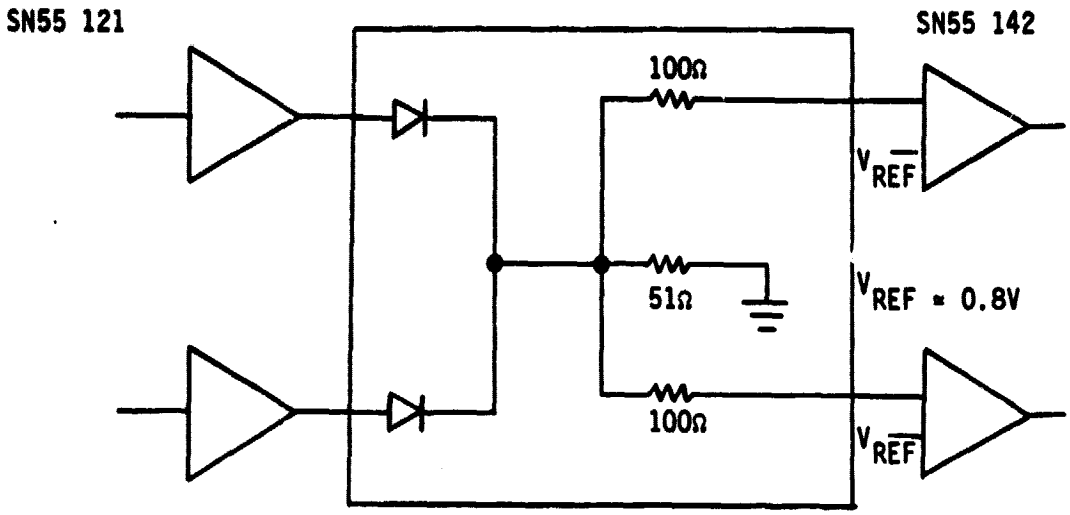


FIGURE 2-7. CROSS-STRAPPING WITH NO SINGLE FAILURE MODES

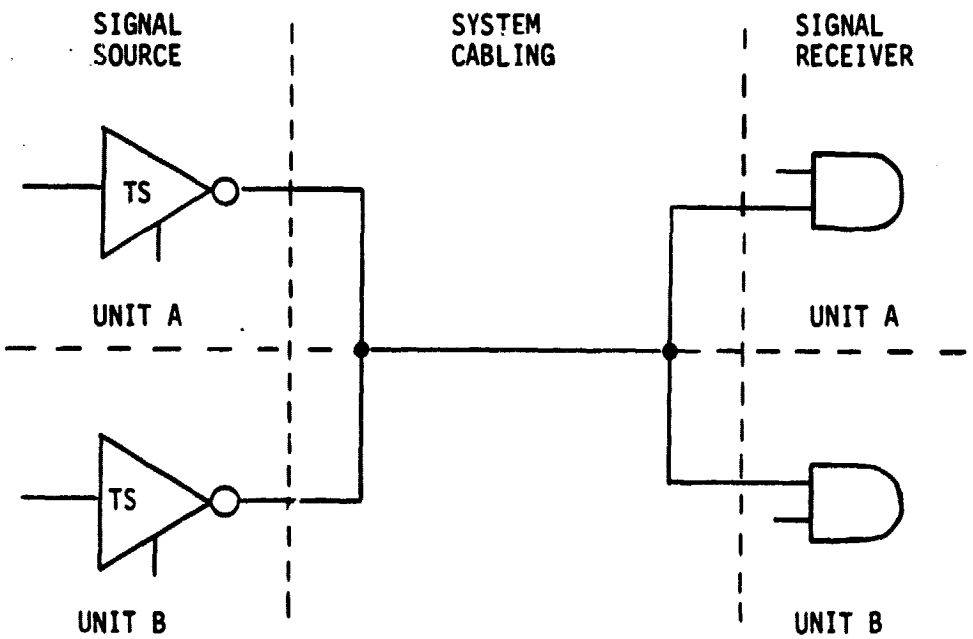


FIGURE 2-8. SIMPLE CROSS-STRAPPING

that in its powered OFF state its output goes to the high impedance (disabled) state.

The impact on systems reliability of the failure (in a shorted mode) of one of the interfaces in Figure 2-8 is negligible. However, the problem of implementing the interface of Figure 2-7 is significant. This study, therefore, recommends the use of the simpler interface since the more complex one can be added later for any application which requires it.

2.4 FTM CONTROL STRATEGY

The fault-tolerant memory subsystem automatically corrects single bit errors. This action is completely transparent to both the computer user and the microprograms. The FTM subsystem does, however, have control functions to control switching of spare memory planes and also to perform testing of the memory. The controls are a combination of hardware and microprogram. The strategy for switching spare planes can have a major impact on the reliability of the unit, however, exact techniques for calculating the effect of switching strategy or reliability are not yet developed.

To understand the significance of the switching strategies, it is necessary first to understand the types of failures, their impact on operation, and their method of detection.

- A "fault" is a hardware malfunction such that the equipment is not capable of doing everything it was designed to do, but it might not be causing any problem at the present time. Example, if a bit is

unable to go to a 1 state but the correct current value is a zero, it isn't causing a problem; but it is still a fault.

- An "error" is the process of the computer getting an improper result at the present time. Example, if a bit read from memory should be a 1 but it is being read as a 0, that is an error.

- A "random fault" in the FTM is one which affects a single bit in a single word in memory.

- A "systematic fault" is either one which affects the same bit in many words or many bits in the same word. The system has been designed, however, to nearly eliminate the ability of a single fault in memory altering the value of more than one bit in a word.

- An "address fault" is a fault which causes a valid word to be stored in and retrieved from the wrong storage location.

- A "transient store" error is where a transient condition in the memory caused a word to be stored with a bad bit in it. However, there is no "hard" fault.

Since the memory can correct single errors, there is not much concern about the existence of words in storage which will give single (correctable) errors when they are read. The most significant problem with single errors is that they have the potential to become double errors which cannot be corrected by the translator

hardware. Random faults are the most probable failure type in a semiconductor memory and there is a low probability that two random failures will occur in the same word. A memory containing n words has a probability of about .5 that there is a double failure after n failures. For n=8K that is about 90 failures and for n=64K it is about 256 failures. Therefore, random failures are predicted to be a problem only if a very large number of them have occurred.

Three strategies will be considered for switching out bad hardware, to restore system operation or reestablish a high level of redundancy. Each approach and its relative merits are discussed below.

STRATEGY A

Reconfigure the system to eliminate the failed bit-plane every time an error is encountered.

Advantages: This is a simple concept which could be easily implemented in hardware without any modification of the current CPU/Memory interface.

Disadvantages: This strategy wastes spare plane usage on random errors which are amply taken care of by the basic correction code. When spare planes are needed for systematic faults ,they will not be available.

STRATEGY B

Reconfigure to switch out a bit-plane whenever it has been determined, during normal operation, to contain a predetermined number of faults, or it is the one which has the most faults whenever a word containing a double error is encountered.

Advantages: This strategy focuses on the distinction between systematic faults and random faults, and would significantly enhance the long term mission reliability.

Disadvantages: Interruption of the operational program to log the location of errors during program execution would have a significant impact on computer operation. There is also a risk that areas of the memory infrequently used because of mission structure could accumulate multiple errors in words which, when they are needed, cannot be practically reconstructed with the diagnostic decoding algorithms, thus impacting the mission. Some memory performance capacity would necessarily have to be allocated to the error logging operation, especially if errors occur in frequently used programs such as vehicle control loops.

STRATEGY C

Rely on the ECC capability during normal program execution, then revert to a test mode whenever a double error is encountered in a word, and at periodic intervals. The test mode would utilize the diagnostic decoding algorithms and properties of the error code to locate and log faults, verify that single errors signaled by the translator are not triple errors, and provide the data for the reconfiguration decision. Reconfiguration would only be performed if a double fault or a systematic fault was detected.

Advantages: This strategy has minimum impact on the operating program yet it utilizes the powerful diagnostic decoding techniques to effectively attain the full potentials of the bit-plane switching capability. All memory locations would be tested frequently, thus minimizing the likelihood of accumulations of many errors in any word before detection of the errors.

Disadvantages: It is difficult to know how often to go into a self test mode, however, this can be programmer controlled rather than "built-in" to the system either in hardware or microcode.

SELECTION: Strategy A clearly does not provide an effective use of the tremendous potential of the spare planes. Strategy B makes a major improvement in the effectiveness of the spare planes, but Strategy C makes several improvements over that of B. Since both B and C require a significant amount of microprogram support, there seems to be little difference in the cost of implementation. Strategy C is selected. The microprogram to support the strategy is described in Section 3.1.2 and the FTM hardware is described in 3.1.1.

2.5 STORAGE ADDRESS EXPANSION

The basic HTC computer calculates storage addresses using 16-bit arithmetic. Since the byte is the basic element addressed in the computer, this results in 2^{16}=65,536 bytes maximum. Applications which require more storage than this must either manage the movement of data and programs in and out of the computer or provide some means to expand memory beyond 64K bytes (K=1024).

Memory expansion involves several facets:

- Generating and holding addresses beyond 16 bits.
- Decoding the most significant bits of the address to form "page select" signals.

- o Driving the signals going to the additional memory.
- o Housing the additional memory.
- o Powering the additional memory.

The first item was the primary subject of the study, whereas the other items were resolved during the implementation phase.

Memory addressing is from three basic sources: operant addresses calculated within the CPU, next instruction addresses taken from the program counter, and storage addresses taken from a device over the direct memory address (DMA) interface. In the HTC all address paths, registers and calculations are 16 bits. If the memory is to be expanded beyond 64K bytes, the maximum memory size must be determined. Discussions with MSFC personnel identified that most foreseeable requirements could be met with 18-bit addressing (256K bytes) and that 20-bit (1 M bytes) would certainly statisfy all requirements.

The next primary issue is to establish the basic approach to the addressing of memory. Both aerospace and commercial computers have been successfully applied using a "sectored" memory where the CPU, I/O, etc. never has access to the full memory at one time. This approach usually involves one or more sector registers to hold the most significant part of the address while the CPU manipulates only the least significant part of the address. With sectored memory the most significant directly controllable address bit usually identifies whether the lowest sector or the "current" sector is being referenced. Sectoring is illustrated in Figure 2-9 assuming the computer has a basic addressing capability of

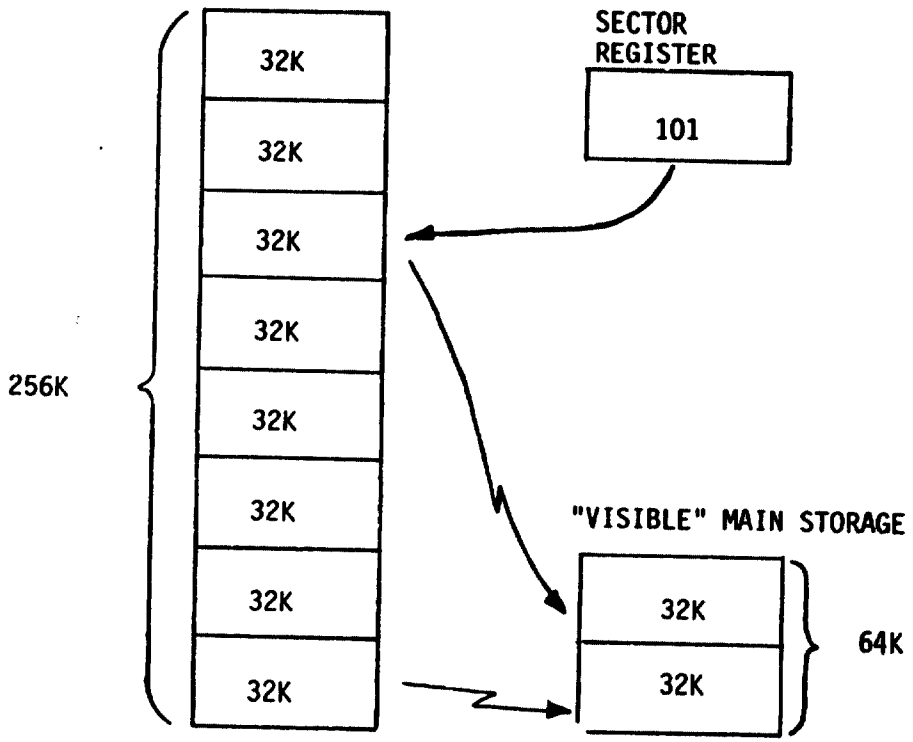


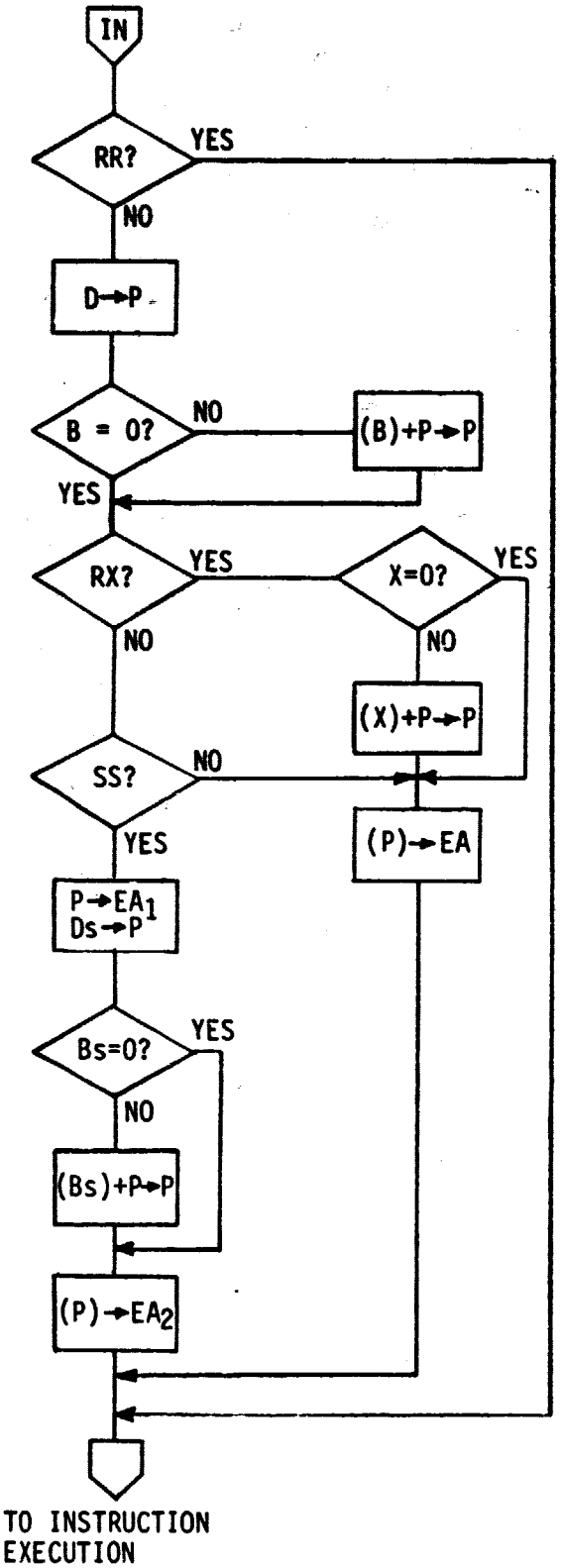
FIGURE 2-9. EXTENDED MEMORY SEGMENTING

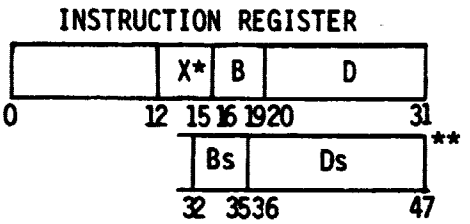
16 bits. The 15 least significant bits (LSBs) identify which byte, of the 32K bytes in a sector, is being addressed. If the most significant address bit (MSB) is a zero, the lowest sector is implied. If the MSB is a one, the value stored in the sector register is used to identify the desired sector. Thus continued access is provided to the low sector of memory, and one other 32K-byte sector can be selected.

The advantage of sectored addressing is that it minimizes the amount of special addressing hardware required. There are, however, some disadvantages with this approach to address:

- The technique is not S/360 compatible.
- Special instructions are required for loading and storing the sector values.
- The programmer must be concerned with memory management as well as application programming.
- If a program or data file resides in two different sectors, many changes in the sector register might be required.
- The MOVE instruction cannot be used to move across sector boundaries except sector zero.

If sectored addressing is not to be used, address computations must involve the full storage address (18 or 20 bits). Figure 2-10 shows a symbolic flow chart of the S/360 address calculation. Provision must be made to provide all the identified additions in 18





* RX FORMAT ONLY
** SS FORMAT ONLY

NOTES: B AND X ARE THE NUMBERS OF THE BASE AND INDEX REGISTERS BUT (B) AND (X) ARE THEIR CONTENTS.

P IS ANY CONVENIENT WORKING REGISTER.

FIGURE 2-10. FLOW DIAGRAM OF THE S/360 EFFECTIVE ADDRESS CALCULATION

or 20 bit arithmetic. There are three approaches to providing 18/20-bit addressing:

- Use the HTC data path as-is and use double precision calculations (two passes through the ALU) to get the addresses.

- Expand the data path to 32 bits for greater arithmetic performance and get the extended addressing free in the process.

- Extend the data path two or four bits for address calculations only.

The double precision microprogramming of the effective address calculation involves the least hardware but reduces machine performance by stretching execution of all but the register-to-register (RR) instructions. Going to a 32-bit data path is a major change in implementation, increases power and weight, and cannot be justified for the sole purpose of extending memory addressing.

Regardless of the approach to address expansion, the following changes to the HTC must be implemented. See the HTC Block Diagram of Figure 2-11.

- The program counter (PC) must be extended either as a counter (preferred) or with a sector register (poot design).

- The CPU's storage address register (SAR) must be expanded.

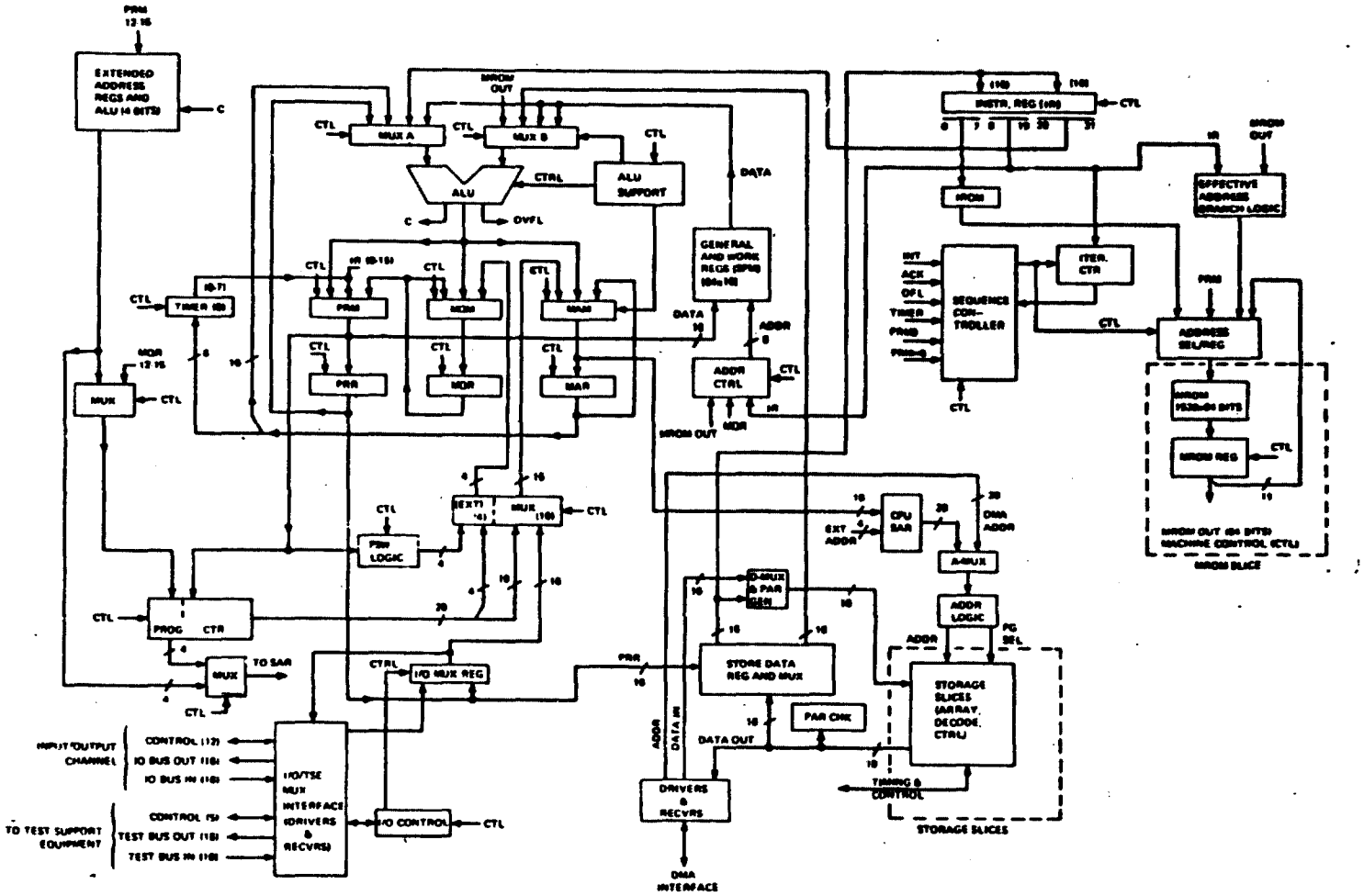


Figure 2-11. SUMC-IIC Block Diagram

- The address multiplexer combining CPU and DMA addresses should be expanded.

- The DMA address interface should be expanded, (if the DMA is to have full access to memory).

- A path must be provided from the PC to the data flow (for store PSW) and to the SAR for I-FETCH.

The implementation requirements of the various approaches to memory extension are illustrated in Figures 2-12, 2-13 and 2-14. In those illustrations, it can be seen that sectored memory provides minimum hardware impact but not significantly less than microprogrammed addressing. Figure 2-12 assumes the PC to remain at 16 bits and share the same sector register as the operants of the effective address calculation. Additional sector registers can be provided for separate sectoring of the program counter, second operant, and DMA. The additional hardware, however, does not seem warranted in an approach with such inherent limitations.

Comparison of Figures 2-13 and 2-14 shows that providing and controlling 2-4 extra bits of ALU and General register is the significant difference between hardware and micropogramming addressing.

Table 2-2 shows a summary comparison between the three approaches. The 32-bit data path approach was eliminated because of the extreme impact both on production hardware (an extra slice) and on development cost. The extended calculation approaches are shown with four bits per flat pack. The sectored memory, however, is shown with extension to only 19-bits. The larger the memory the more problems encountered (with sectored memory) so 18-bit addressing (256K-bytes)

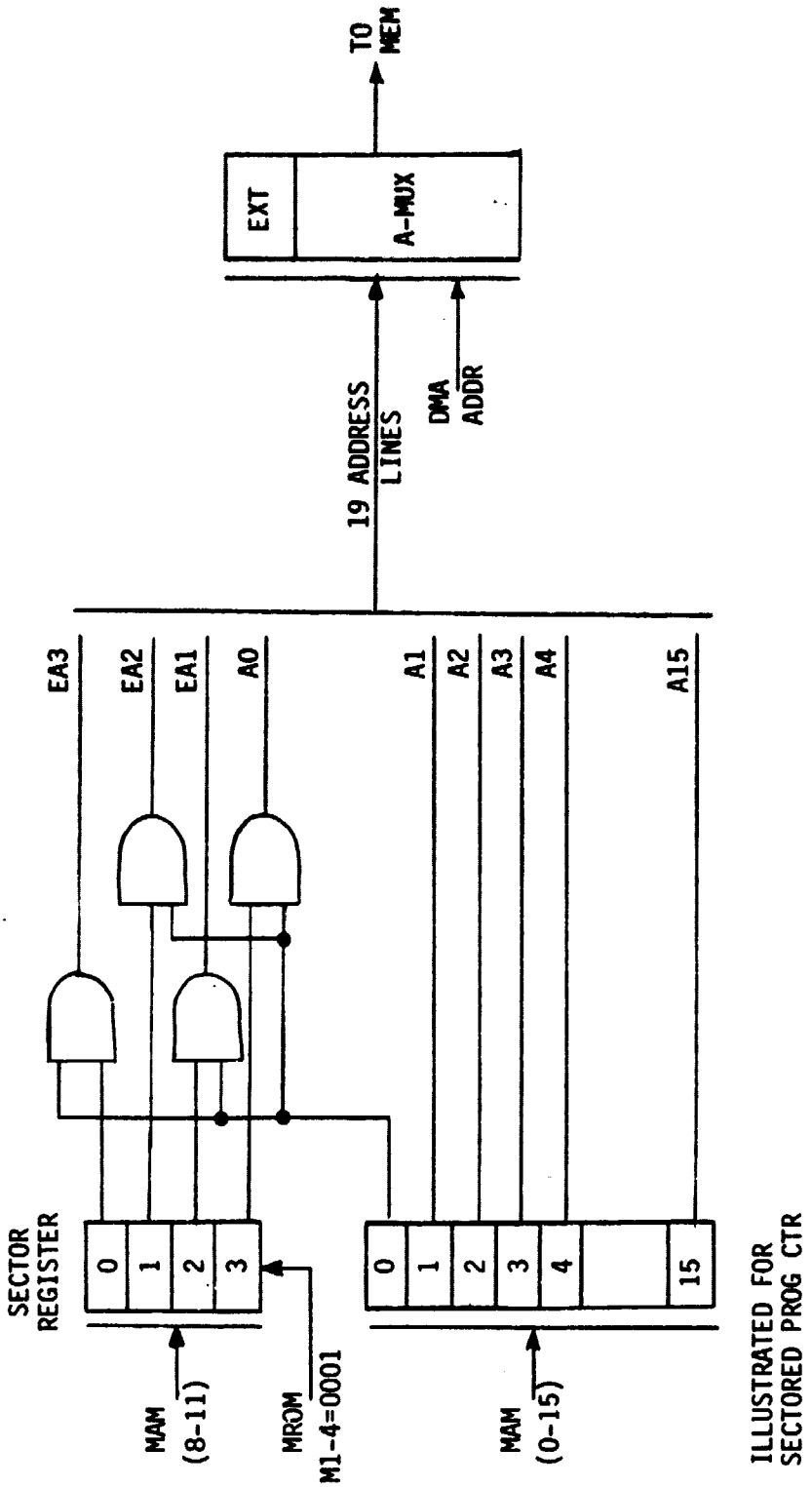


FIGURE 2-12. SECTORED MEMORY IMPLEMENTATION

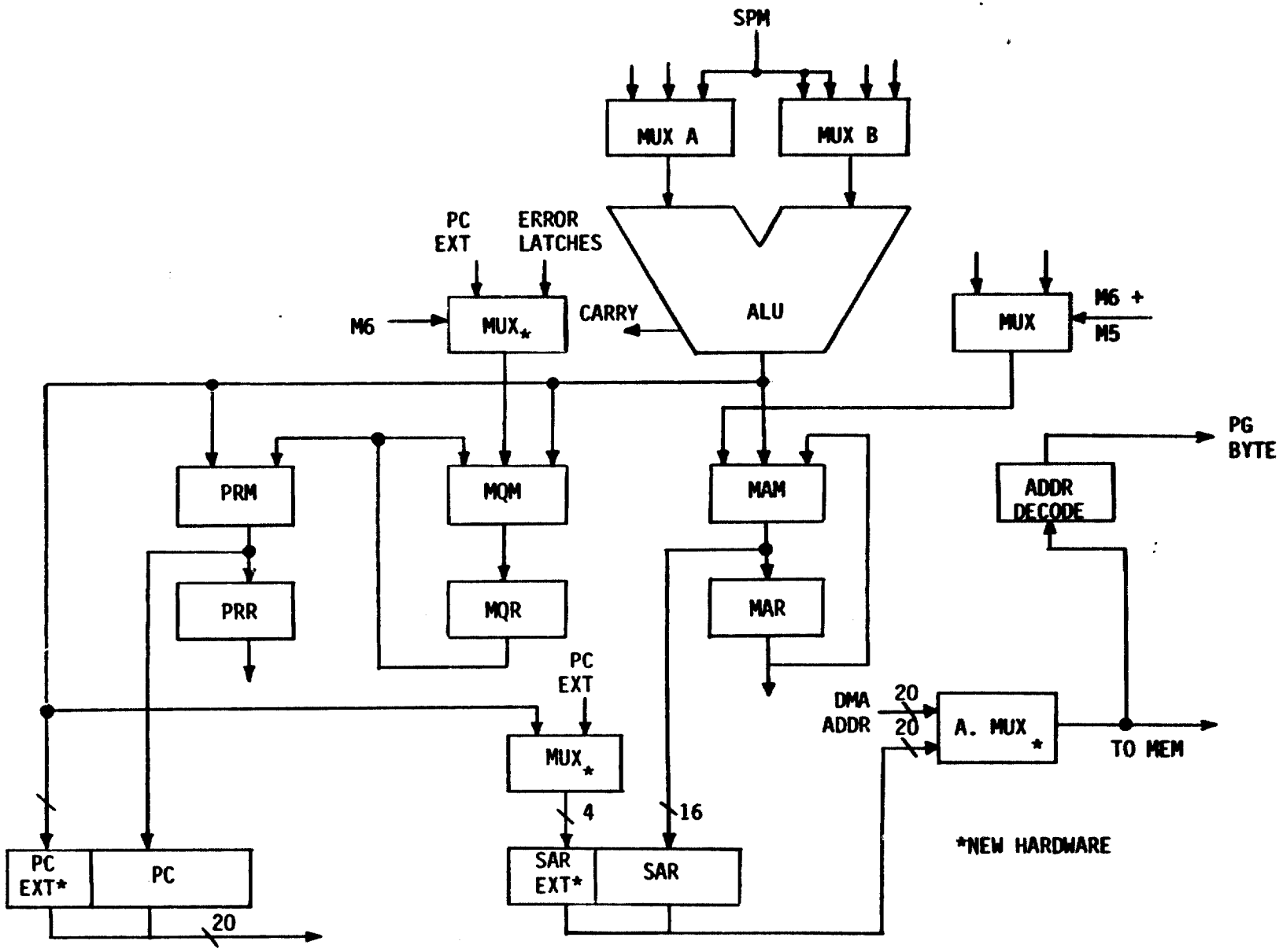


FIGURE 2-13. DOUBLE PRECISION EA MICROPROGRAM HARDWARE

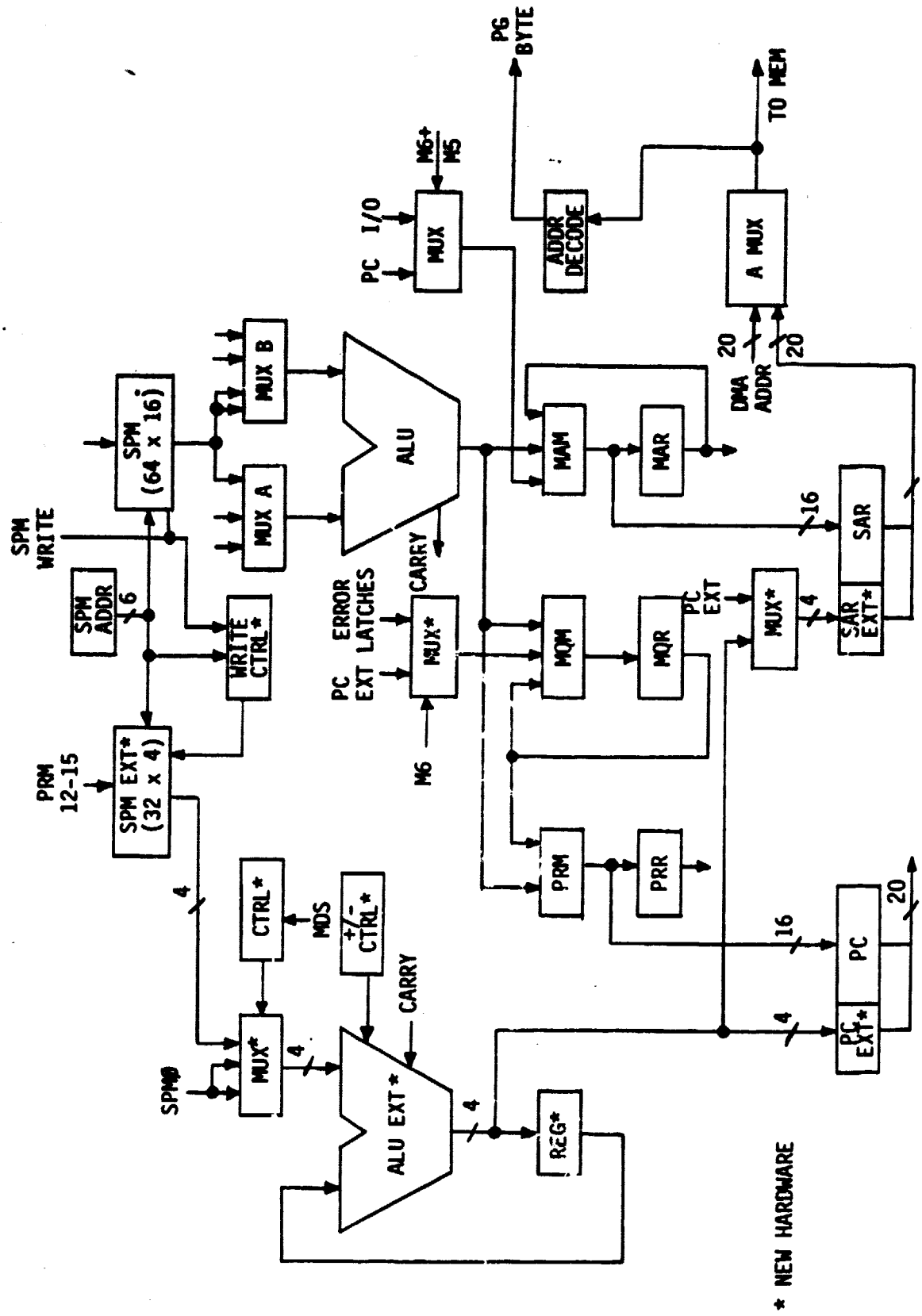


FIGURE 2-14. DATA PATH EXTENSION

Table 2-2. Summary of Address Expansion Trades

TRADE STUDY PARAMETER	CANDIDATE TECHNIQUE	SECTORED MEMORY, 19-BIT ADDR.	U-CODE EA CALCULATION, 20-BIT ADDR.	EXPANDED ALU PATH, 20-BIT ADDR.
Estimated Hardware		7 FPs	8 FPs	19 FPs
System/360 Compatible ?		No	Yes	Yes
Microcode Required		Modest	Modest	Modest
Memory Management Required ?		Yes	No	No
Reduce Performance ?		Slightly	Yes	No

would probably not be exceeded. Also 19-bits represents an efficient use of hardware. The hardware estimates in the table are those which were believed at the time of the trade study.

The problems associated with the management of a sectored memory plus the loss of S/360 compatibility makes sectored memory very unattractive. The expanded ALU/DATA Path approach was selected over the microprogrammed approach to get a 10% increase in performance at the cost of a few flat packs. Section 3.2 of this report shows the implementation of the address expansion.

3.0 IMPLEMENTATION

3.1 FAULT-TOLERANT MEMORY

The Fault-Tolerant Memory (FTM) is a storage system which tolerates single and multiple errors within words read from memory. The FTM system is comprised of three major segments: Storage Array, Translator, and Error Correction Algorithms.

3.1.1 Storage Array

The storage array is composed of basic memory modules (BMMs) which are hermetically sealed and each contains 8192 bits of N-channel FET random access storage organized as an 8K x 1 bit array with on the chip address decoding. Each memory module is part of a bit plane. There are 16 bit planes corresponding to the 16 bits to be stored from the CPU, 6 check bit planes and 4 spare bit planes. A bit plane provides one bit to the word read from storage. The bit plane organization ensures that any failure in a BMM will affect at most one bit in any word read from storage. This feature significantly enhances the effectiveness of the error correction code. Any failure in a module can mutate only one bit in any given word stored. Figure 3-1 contrasts the storage array organization for 64K simplex and Fault-tolerant machines.

3.1.2 Translator

Referring to Figure 3-2, the Translator is functionally partitioned into six major data flow areas: a storage data register (SDR) which

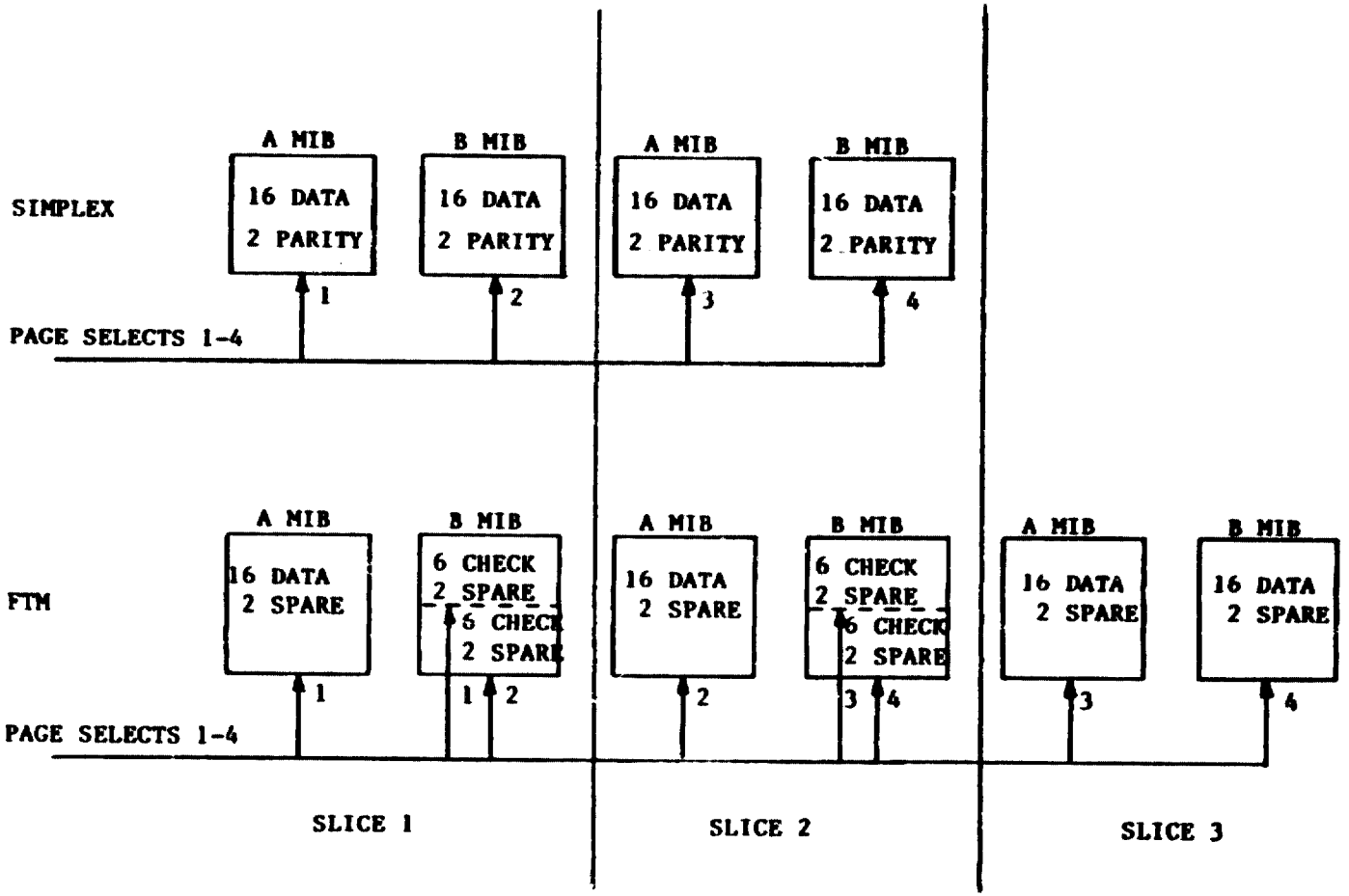


Figure 3-1. STORAGE ARRAY ORGANIZATION

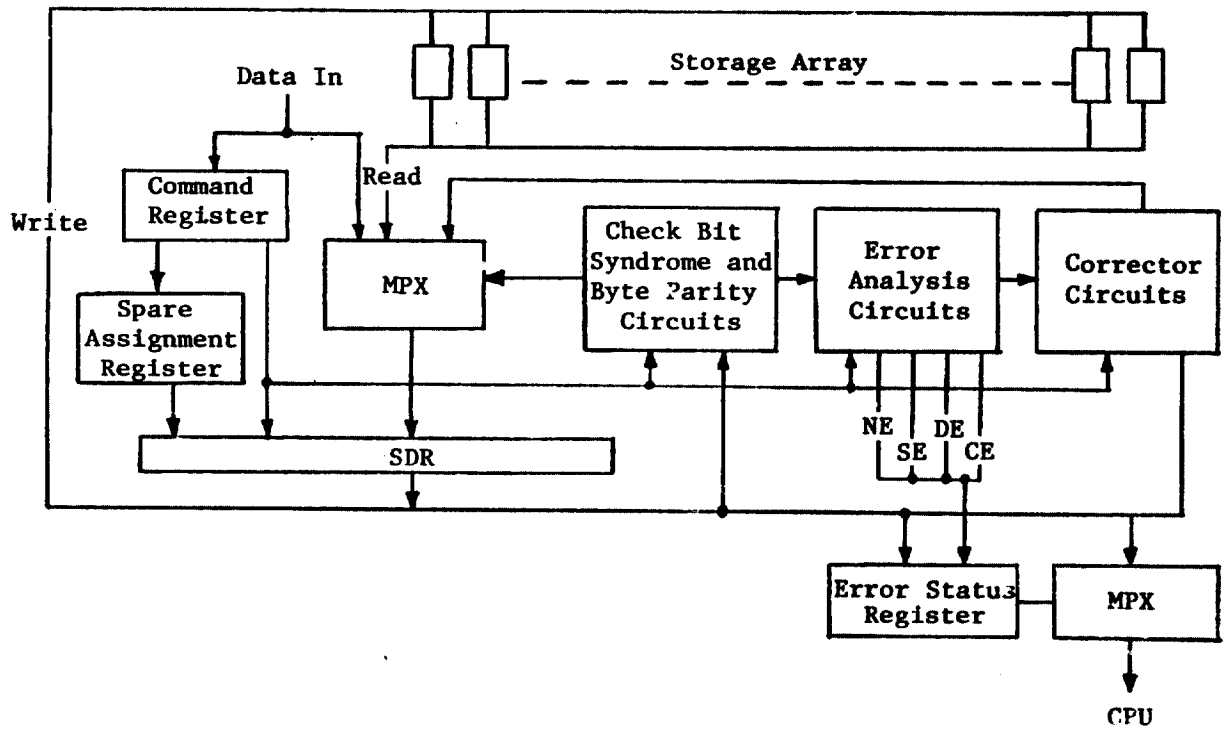


Figure 3-2. TRANSLATOR FUNCTIONAL BLOCK DIAGRAM

includes an input multiplexer, parity trees, corrector, error analysis, command and status registers, and spare assignment register. The SDR is the major working register for the translator. All data inputs to be stored from the CPU are read into the SDR and all data read from the storage array is read into the SDR.

3.1.2.1 Parity Trees

Parity trees in the translator are used for four purposes: generating check bits on store operations, checking byte parity bits on store operations, generating syndromes on read operations, and self-testing of translator circuits.

3.1.2.1.1 Check Bit Matrix

Referring to Figure 3-3, there are 16 data bit positions labeled 1 through 16. There are six check bit columns labeled C1-C6. Each check bit is generated by parity trees to give odd parity over the field consisting of itself and eight associated data bits in the same row of the matrix. Thus, C1 could be generated as zero or one if data bits 1 through 8 had odd or even parity, respectively. Similarly, C2 would be generated to give odd parity over the field consisting of itself and data bits 6 through 13. It should be noted that each column of the parity check matrix consists of an odd number of 1's. The data bit columns have three 1's and the check bit columns have a single 1. This constitutes a Hamming code of odd-weight.

Bit Position

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	C1	C2	C3	C4	C5	C6
S1	1	1	1	1	1	1	1	1									1					
S2					1	1	1	1	1	1	1	1	1					1				
S3	1		1		1	1			1		1				1	1			1			
S4	1			1			1		1			1	1	1		1				1		
S5		1		1	1			1		1		1		1	1						1	
S6		1	1							1	1		1	1	1	1						1

Figure 3-3 Parity Check Matrix for 16 Data Bits

On store operations, the 16-bit word from the CPU is first stored in the SDR. The 16-bit word is then flushed through the parity trees to generate check bits which are also stored in the SDR. Subsequently, a 22-bit word is transmitted to the storage array (16 data bits and 6 check bits).

3.1.2.1.2 Error Detection and Location

On read operations, each of six fields, consisting of eight data bits and an associated check bit, is checked for odd parity. The parity indication signals generated for these six 9-bit fields are called syndromes labeled S1 - S6 in Figure 3-3. In the event that one or more syndromes indicate a discrepancy, an error is flagged. The pattern of the syndromes is analyzed to determine the type of error and, in the event of a single error, the syndrome pattern indicates the position of the errant bit.

Each data bit and each check bit has a unique pattern of 1's in its column. Thus, if data bit 1 was in error, then syndromes 1, 3, and 4 would indicate discrepancies. The combination of syndromes (1, 3, 4) uniquely identifies data bit 1 as the errant bit. In this way, the syndrome patterns are decoded to locate a single-bit error.

3.1.2.1.3 Self-Testing

The construction of the parity trees used in the translator augments the self-checking/self-testing properties of the translator. Figure 3-4 illustrates the organization for one of the three parity trees on a parity chip. There are nine input bits per tree and each tree is divided into a six-bit section and a three-bit section. There is a partial output for the six-bit section of the tree and another for the three-bit section. There is also a combined output which represents the parity over all nine input bits. The pair of partial outputs is called the "morphic" output of the parity network, while the combined output is the usual logical output. Since odd parity is being used, an error-free syndrome from the morphic output is indicated by an 01 or a 10 signal on read cycles. In the event of a fault within the parity tree network which results in an erroneous output, only one leg of the morphic output will be affected. Therefore, single gate failures in these circuits propagate to the output where they may be detected. Thus an odd parity input to a parity circuit, containing an error causing fault, will result in an 00 or 11 morphic output. Of course, an even parity input to a fault-free parity tree, will also cause the morphic output to be 00 or 11.

3.1.2.2 Corrector

The corrector consists of a correction decoder and exclusive OR gates which are in line with each of the 22 bits. Correction occurs as data is being transmitted to the CPU. Correction is complete before the CPU receives a read data ready indication. The syndromes generated by the parity trees are decoded into 22 bits. The correction decoder illustrated in Figure 3-5 consists functionally of 27 six-input AND gates which decode each of the 20

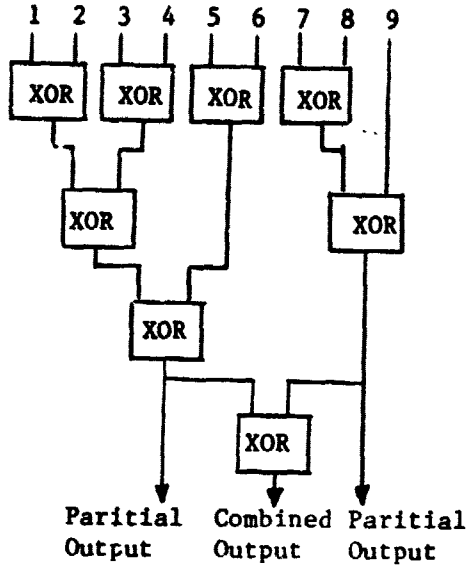


Figure 3-4. FUNCTIONAL PARITY TREE REPRESENTATION

combinations of six things taken three at a time, the six combinations of six things taken one at a time, and the single combination of none of six. The outputs from the decoder are wired to the appropriate SDR bit positions. All twenty of the three of six combinations are available on the chip; however, only the appropriate 16 are utilized for the sixteen data bit positions in the code chosen. The inputs to the decoder are the combined outputs of the six syndrome parity trees.

The outputs of the correction decoder are then exclusively ORed bit-by-bit with the data as read from the storage array. Whenever the two inputs disagree, correction occurs.

3.1.2.3 Error Analysis

The error analysis portion of the translator is perhaps the most unique portion. It is implemented in morphic logic. The morphic logic uses dual line pairs to replace the single lines in conventional logic gates arranged as two independent tree structures so

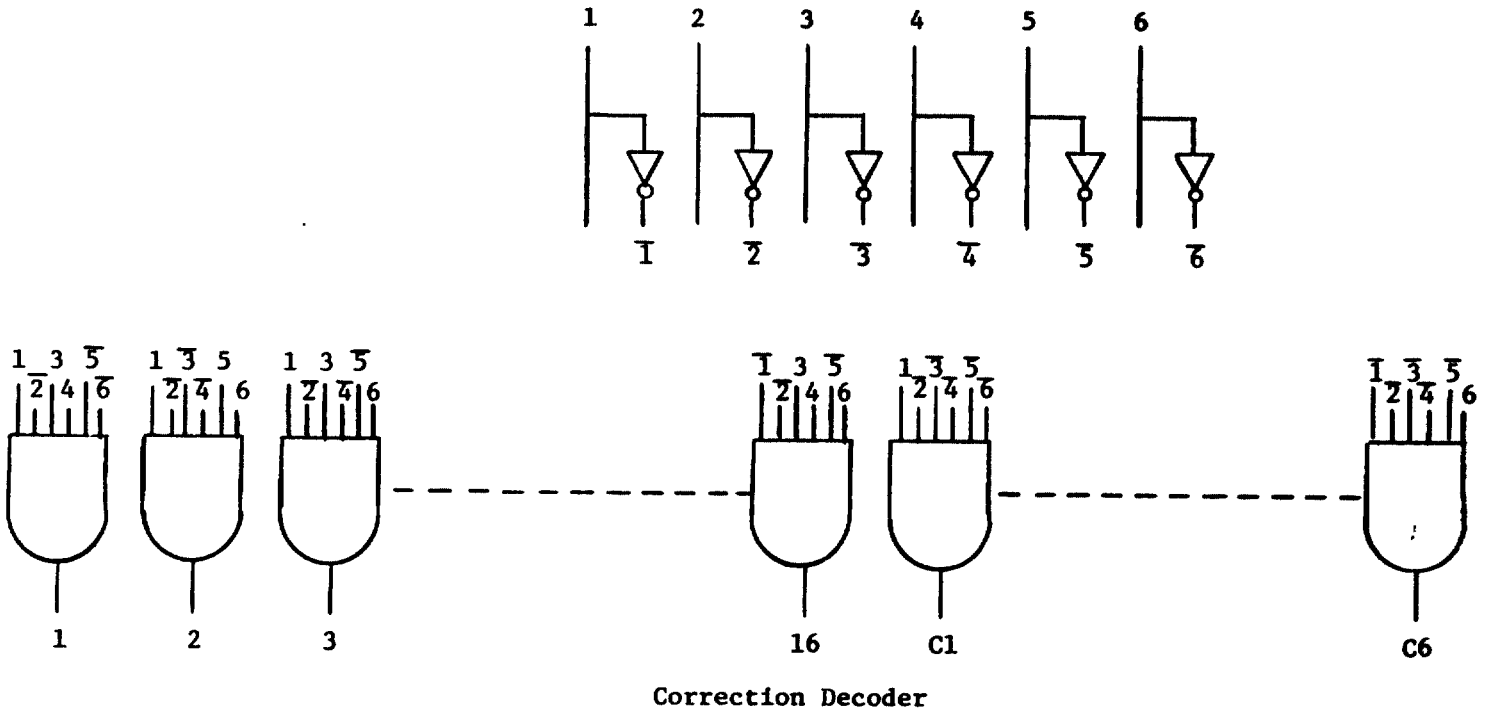


Figure 3-5. CORRECTION DECODER

that a fault of a single gate in the morphic logic propagates to the output where it can be detected. Circuits representing morphic invert, morphic AND, morphic OR, morphic exclusive-OR, etc., have been devised. Combinations of these morphic gates can be utilized to implement any logical function. The morphic logic equivalent of a conventional logic 1 is a 01 or a 10, $\begin{matrix}[01]\\[10]\end{matrix}$, on the line pair. The morphic logic equivalent of a conventional logic 0 is $\begin{matrix}[00]\\[11]\end{matrix}$ on the morphic line pair. For explanation of the error analysis circuits for this translator, the nomenclature 1_M $\begin{matrix}[01]\\[10]\end{matrix}$, and 0_M $\begin{matrix}[00]\\[11]\end{matrix}$ will be used.

The translator error analysis will be illustrated by explaining its operation for checking the word read from storage. The verification of byte parity checking and generation and check bit generation will then be explained.

The AND_M whose output is labeled A in Figure 3-6 has inputs from the morphic output of the syndrome generation parity trees S1 - S6. Since odd parity is used in the encoding, on read-out all of the syndrome partial signals should be 1_M if no error has occurred. Thus, the output A should be 1_M. Two parity trees are shown as the input B in Figure 3-6. There are an even number of syndromes (6). One each of the two morphic lines from each of the syndrome generators (the byte parity tree inputs are inhibited during read cycles) are inputs to the two parity trees whose outputs form B. Since the syndrome no error condition is 1_M and there are overall an even number of syndromes, there should be in total an even number of morphic (and logical) 1's under a no-error condition.

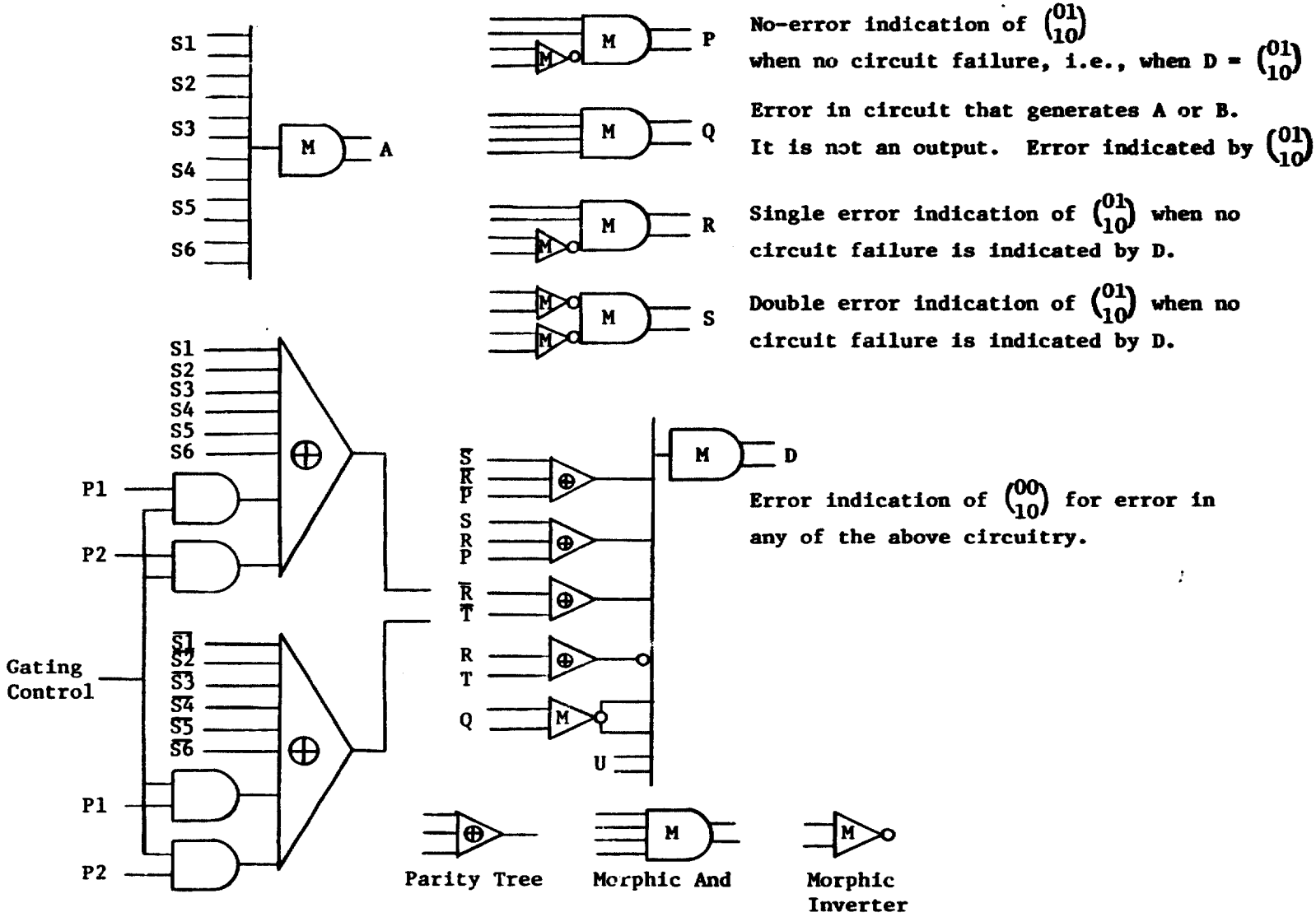


Figure 3-6. ERROR ANALYSIS

Since this is true, both parity trees should have like parity either odd or even since the sum of two odds or of two evens is even. The input B in Figure 3-6 should be 0_M under a no-error condition. The AND_M gate whose output is P indicates NO ERROR as 1_M when the A and B signals are normal.

The AND_M gate whose output is Q indicates a translator circuit failure condition. There is no valid condition of the inputs which causes outputs A and B to be 1_M simultaneously. This condition is indicative of a failure in the circuits which generate A or B. Therefore, the AND_M gate whose output is Q senses this condition as a circuit error.

A single error is mainfest as an odd number of 0_M syndromes -- one syndrome or three syndromes having a value of 0_M. Under this condition, the output A will be a 0_M and the output B will be 1_M. The output A is inverted to make it a 1_M and combined with the output B which will be 1_M to cause signal R to be 1_M -- the single-error condition signal.

The AND_M gate whose output is S senses a double-error condition. The output A will be 0_M as will the output B in the presence of a double error in the word read from storage. Inversion of both these outputs makes them both 1_M and when combined in the AND_M gate, whose output is S, indicates the double-error condition.

The AND_M whose output is D indicates a circuit-error condition. The input U is for byte parity circuit checks. The signal Q (mentioned previously) is inverted because its normal (no error) indication is an 0_M. In order to maintain consistency, all inputs to the AND_M should, under normal conditions, be 1_M's.

Therefore, the signal Q inverted is 1_M during normal operation. The signal T is used for checking the validity of the generated check bits on write operations and the validity of the generated byte parity bits on read operations. It is proved that, with the code structure herein utilized, the parity of the byte parity bits and the parity of the code check bits should be the same; therefore, their combined parity should always be even. The signal T is the morphic output of a parity tree whose inputs are the two byte parity bits and the six combined outputs of the parity trees which generate the check bits.

The two parity trees with T and R inputs together with an inverter perform the logical operation T = R which is true when there is a valid single error condition and no circuit failures. Certain circuit failures might be detected as a single data error without this check.

A check is made to see that there is not an even number of the inputs P, R, and S in a 1_M state because P, R, and S are mutually exclusive conditions. Should none or two of these three signals be up, there will be an even number of logical 1's which, distributed between the two parity trees whose inputs are PRS, will make their output 0_M. That is a failure indication causing the output D to be 0_M.

This discussion of the read cycle operation is intended to illustrate how the morphic logic is utilized to provide self-checking during normal operation. Since the normal data flow constantly changes, the translator circuits assume both states, which provides the self-testing property.

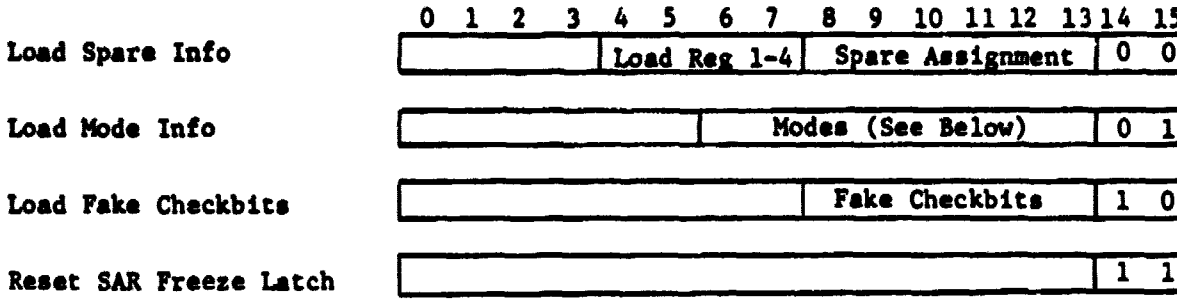
3.1.2.4 Command and Status Registers

Microcode in the NSSC-II is used to communicate with the translator to support fault isolation and correction. Each time the microcode command NO-OP, is executed the contents of the SDR are interpreted as a command to the translator and loaded into the command register. There are four basic types of commands: spare assignment, mode changes, loading fake checkbit register, and reset storage address register (SAR) freeze latch. The spare assignment command is used to load a spare assignment register which substitutes the spare assigned for the bit specified. The load fake checkbit is used when checkbits other than the ones normally generated are to be stored in memory in support of memory diagnostics. The SAR in the CPU is frozen when a double error occurs so the location may be interrogated and the failing bits identified. At the end of this interrogation, the latch prohibiting reloading of the SAR is reset with the reset SAR freeze latch command. Special modes of operation are required to execute the advanced microcoded storage diagnostics employed. These modes are specified to the translator by the mode command. Figure 3-7 shows the command NO-OP structure and gives a brief definition of the various modes.

The status register contains the Translator Error/Status word and is used by the diagnostic decoding algorithms to determine status after an FTM error interrupt and to support multiple error correction. Definition of the Translator Error/Status Word is contained in Figure 3-8.

3.1.2.5 Spare Assignment Register

When the decision is made in microcode to substitute a faulted bit with a spare bit plane, one of four spare assignment registers



Bit	Mode	Function
6	Use Fake Checkbits	Uses bits stored in Fake Checkbit register instead of stored checkbits on read or generated checkbits on writes
7	Inhibit Write Error Checks	Inhibits checking for CPU parity errors or translator errors on write cycles
8	Inhibit Correct	Forces translator timing to not generate the correction clock used in test mode
9	Inhibit Load	Forces translator timing to not generate the SDR load pulse used in test mode
10	Error Check SDR	Causes read cycle to use data in CPU SDR rather than memory data, when used with bit 8; can error check generated word without using memory used in test mode; if used with bit 15, SDR will contain error status of word at end of cycle
11	Reconfigure Mode	Force translator to read from old bit plane (bit being reconned only) and write TDR data to assigned bit plane
12	Test Mode	Forces translator to use TDR rather than exclusive ORs for corrections and allows for additional control
13	Put Errors On Data Bus	Data in bus will contain checkbits and error status

Figure 3-7. COMMAND 'NO-OP' COMMAND STRUCTURE

Bit	Meaning
0	No error detected during a read in test mode.
1	Single data error detected during a read in test mode.
2	Double data error detected during a read in test mode.
3	Translator error detected during test mode.
4-9	Checkbits as read from memory.
10	Translator or CPU parity error detected on write.
11	Spare.
12	Double data error during read in normal mode.
13	Translator error during read in normal mode.
14	Spare.
15	Spare.

Figure 3-8. TRANSLATOR ERROR/STATUS WORD

is loaded with the syndrome pattern of the faulted bit. This syndrome pattern is subsequently decoded and the identification of the faulted bit is provided as input to the SDR. The SDR then switches the faulted bit plane out of the data path and switches the spare bit plane into the data path.

3.1.3 Error Correction Algorithms

Several algorithms, implemented in microcode, permit maximum utilization of the fault tolerant memory feature. A flow diagram of the microcode routines containing these algorithms is shown in Figure 3-9. During program execution, the FTM operates under direct hardware control and corrects single errors as they occur. If an uncorrectable memory error occurs, the microcode routines depicted in Figure 3-9 are invoked which analyze the error and take appropriate actions to reconfigure memory to eliminate the error. As the routines are executed, the FTM system provides status reports to the outside world via the FLAG RMU control signal and a code word placed on the I/O channel output bus. The code words and their meaning are listed in Figure 3-10.

3.1.3.1 Error Location

If a double read error is detected, the address of that word is frozen by the memory interface hardware and the microprogram takes two actions: First, the READ is retried 64 times to determine that it was a hard failure. If any retry results in a correctable error, the normal program execution will be resumed.

If the error was "hard", the microprogram initiates the second action of identifying the "stuck" locations in the word with the

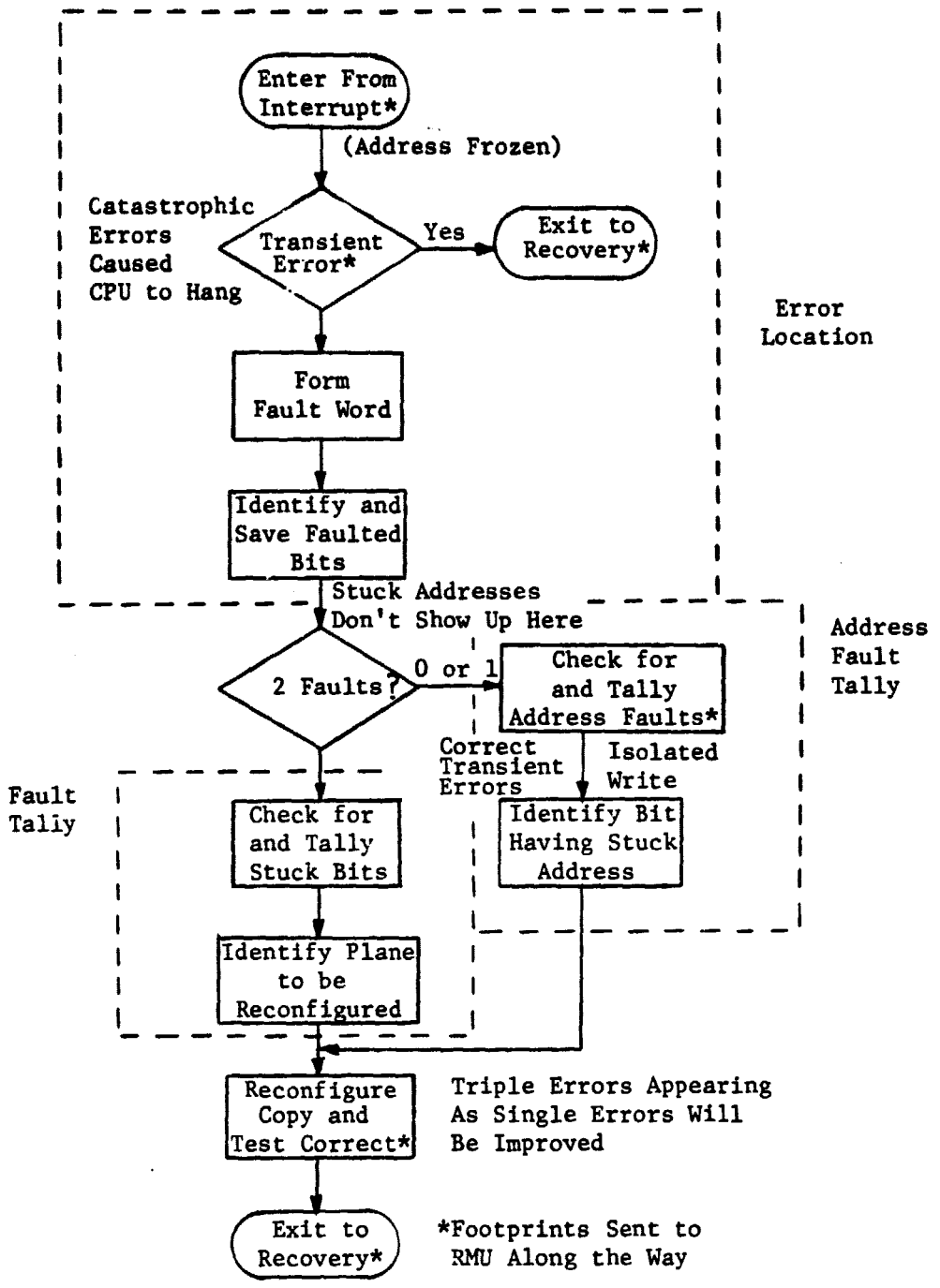


Figure 3-9. DIAGNOSE REPAIR FLOW

- FLAG RMU - Informs the RMU that the NSSC-II has loaded the FTM Status in the I/O Register. Codes listed below appear as bits 8-15 of the I/O Bus Out.

Status Code	
0000 0001	Multiple Error
010N NNNN*	More than three faults during talley
0000 0100	Fault Talley Entered
0001 0000	Address Fault Talley Entered
0000 0010	Reconfiguration Entered
0010 0000	Copy/Test/Correct Entered
0000 0000	Successful FTM Exit
1000 0100*	Translator Error During Read
1000 1000*	Correction Failure
1010 0000*	Translator Error during Write
1000 0010*	Reconfiguration Failure
1000 0000*	Error in Error Detections

* Results in a FTM "Hang" Condition

** N = Number of Faults during talley

FIGURE 3-10. RMU STATUS CODES

routine called Form Fault Word which is flow charted in Figure 3-11.

- Read the data word without correction and place in temporary storage.
- Read the check bits and place in temporary storage.
- Store the bit-by-bit inversion of the data and check bits in the same storage location.
- Reread the data and check bits and compare (exclusive OR) with the original data and check bits.
- Zeros will identify the locations of any bits which are not able to be inverted (stuck at 1 or 0).
- If two or more bits are stuck, the fault location data is stored and the microprogram proceeds to the fault tally segment.
- If less than two bits are stuck, there is an addressing error or a transient WRITE error which caused the storage of a bad bit.
- If less than two stuck bits were found, the analysis proceeds to find stuck addresses in the entire memory (ADDRESS TALLY).

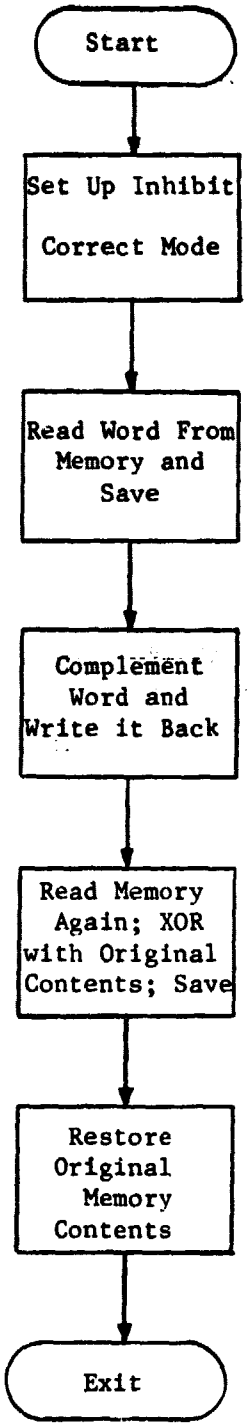


Figure 3-11. FORM FAULT WORD

3.1.3.2 ADDRESS TALLY

ADDRESS TALLY is invoked because a double error condition exists and the fault word routine found less than two faults. This routine is flow charted in Figure 3-12.

- o Stuck addresses are found as follows:
  - Read word A and store it temporarily.
  - Read word B and store it (where the address of B is different from A by a single bit).
  - Complement A and store it in B.
  - Check to see if A changed. If so one bit of B is being stored in A.
- o Each stuck address which is found is tallied for subsequent reconfiguration.
- o If there was a transient WRITE problem, it will be corrected during the final microprogram segment which is TEST/CORRECT.

3.1.3.3 FAULT TALLY

FAULT TALLY is invoked because the fault word routine found two or more faults at the error location. This routine is flow charted in Figure 3-13. The FAULT TALLY routine checks the entire memory for "stuck at" faults and tallies them by bit number. At the

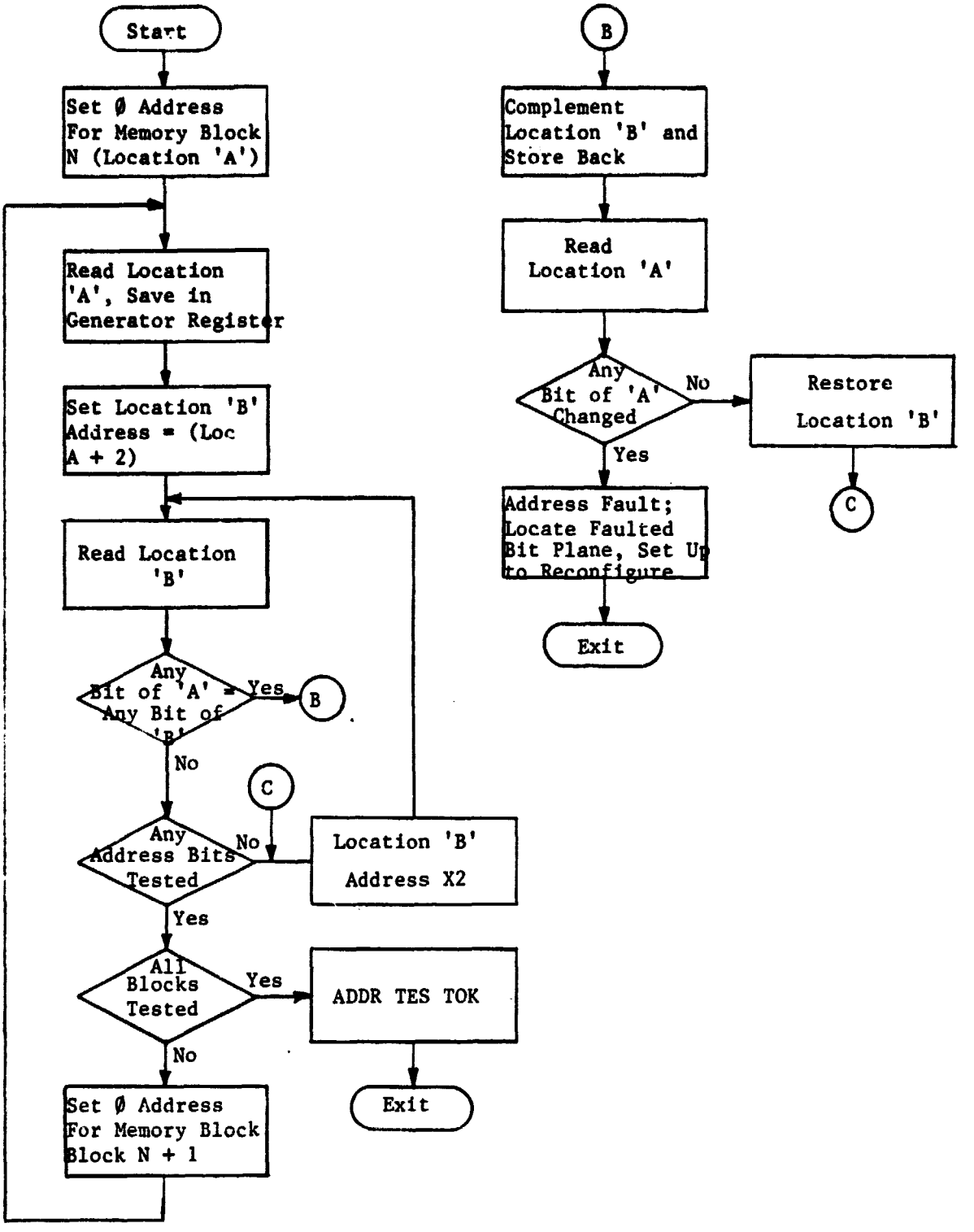


Figure 3-12. ADDRESS FAULT TALLY

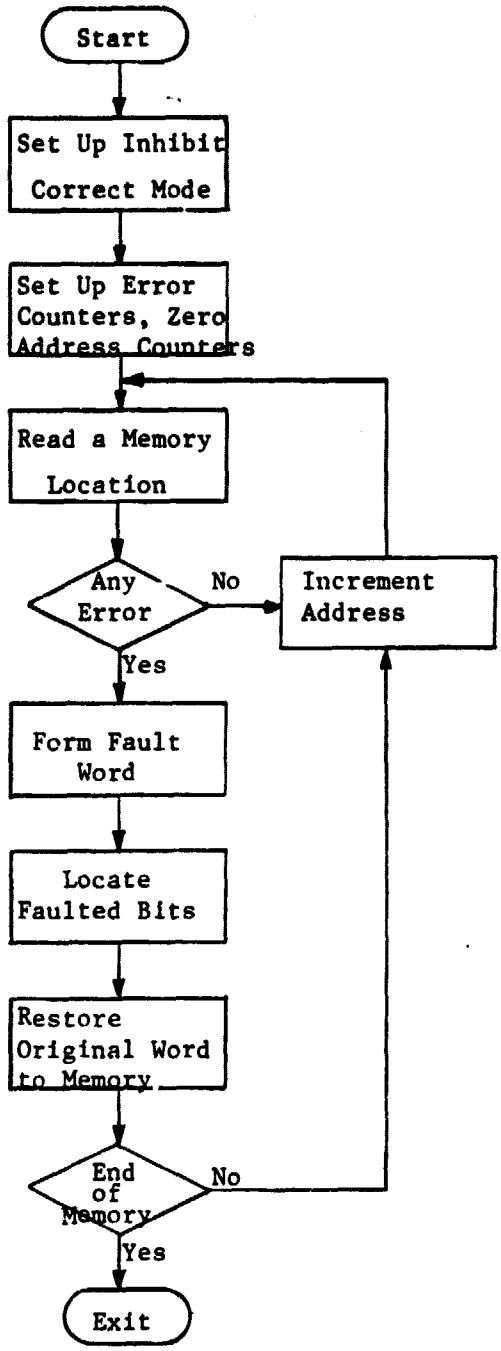


Figure 3-13. FAULT TALLY ROUTINE

conclusion of this routine the counts for each bit location are evaluated and a reconfiguration decision is made. The decision logic is as follows:

- o If a double error caused the diagnostics to be run, a plane will be substituted in the RECONFIGURE routine as follows:
  - If ERROR LOCATION found two or more stuck at bits the one substituted will be the one with the largest number of faults detected by FAULT TALLY.
  - If there was only one stuck at fault and an addressing fault the address fault will be substituted.
  - If there was one stuck bit and a transient WRITE the stuck bit will be substituted.
- o If the analysis microprogram entered via diagnose instruction, the only reason for a plane substitution is if the tally of one or more bits exceeded the threshold of 512 faults. Since each chip contains 2048 bits this threshold seems reasonable for a systematic fault. A different threshold can be substituted by burning-in a different value in the microprogram memory.

The tally operation is essentially the same as the fault location operation, as it looks for stuck bits by the READ/INVERT/STORE/READ/COMPARE sequence. This tally operation can be performed on the entire memory at one time or can be broken into segments.

3.1.3.4 RECONFIGURE

The RECONFIGURE routine is the short microprogram sequence which "tells" the translator which spare plane number should be substituted for which data or check bit. If the bad bit plane is a spare bit plane, it must be unassigned to permit the new spare bit plane to be effective. This routine is flow charted in Figure 3-14.

3.1.3.5 TEST/COPY/CORRECT

The TEST/COPY/CORRECT routine has two functions: copy the information from the old plane to the new plane and correct all possible errors. This routine works with each location in memory in sequential order reading from the old plane and writing to the new plane. This routine is flow charted in Figure 3-15.

Each time a READ operation results in any error, a special correction routine is used to correct the data. This proprietary routine is so constructed that it can form the correct word for all single, double, and triple "stuck at" type faults and one "soft-error" in combination with zero, one, or two "stuck at" type faults. This routine is flow charted in Figure 3-16.

3.2 FTM SYSTEM MANAGEMENT

The previous sections presented the implementation of the Fault Tolerant Memory hardware and microcode. The benefits derived from this system implementation are: (1) the capability to recover from permanent and transient type memory errors, (2) extending the

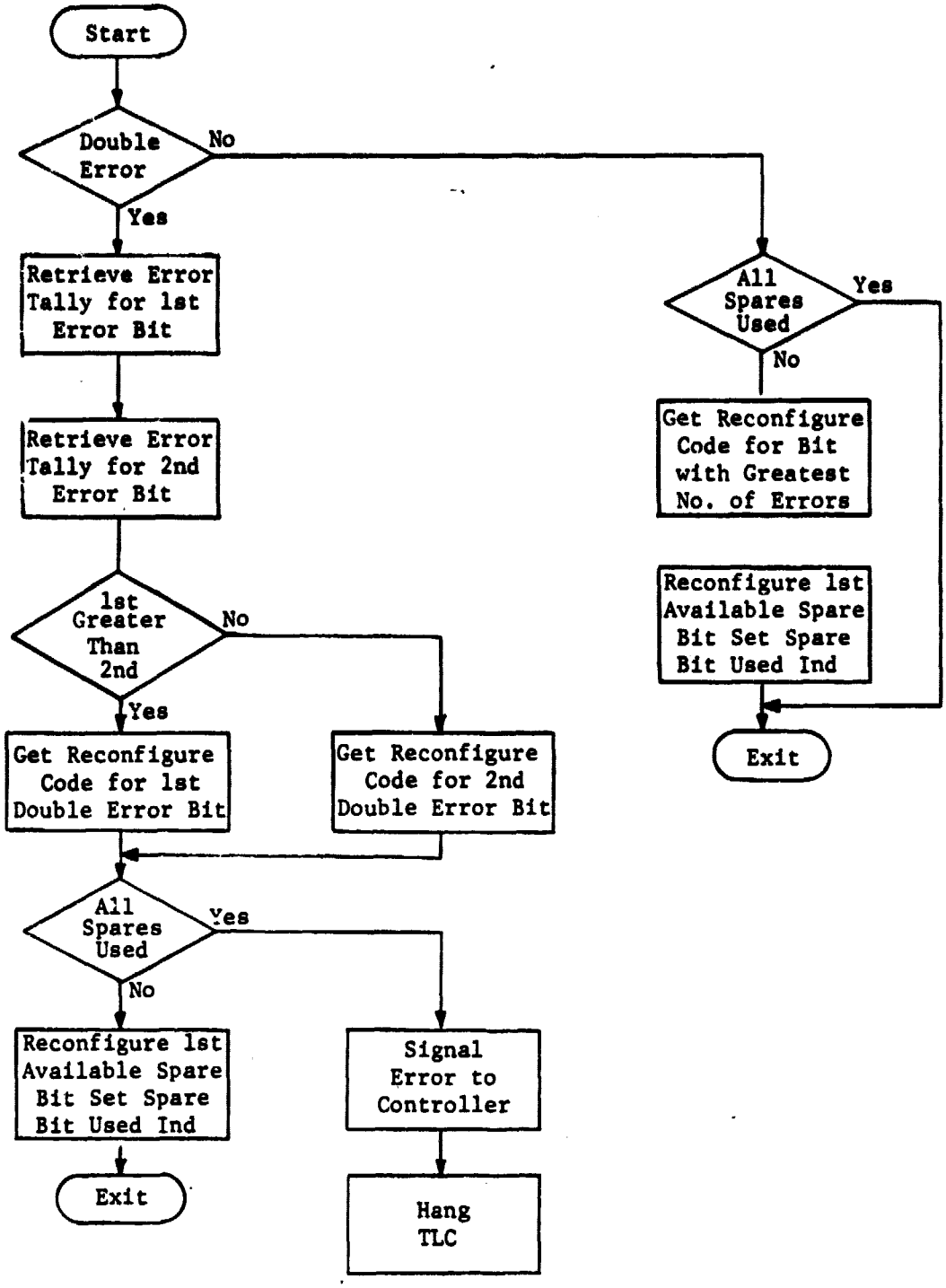


Figure 3-14. RECONFIGURE ROUTINE

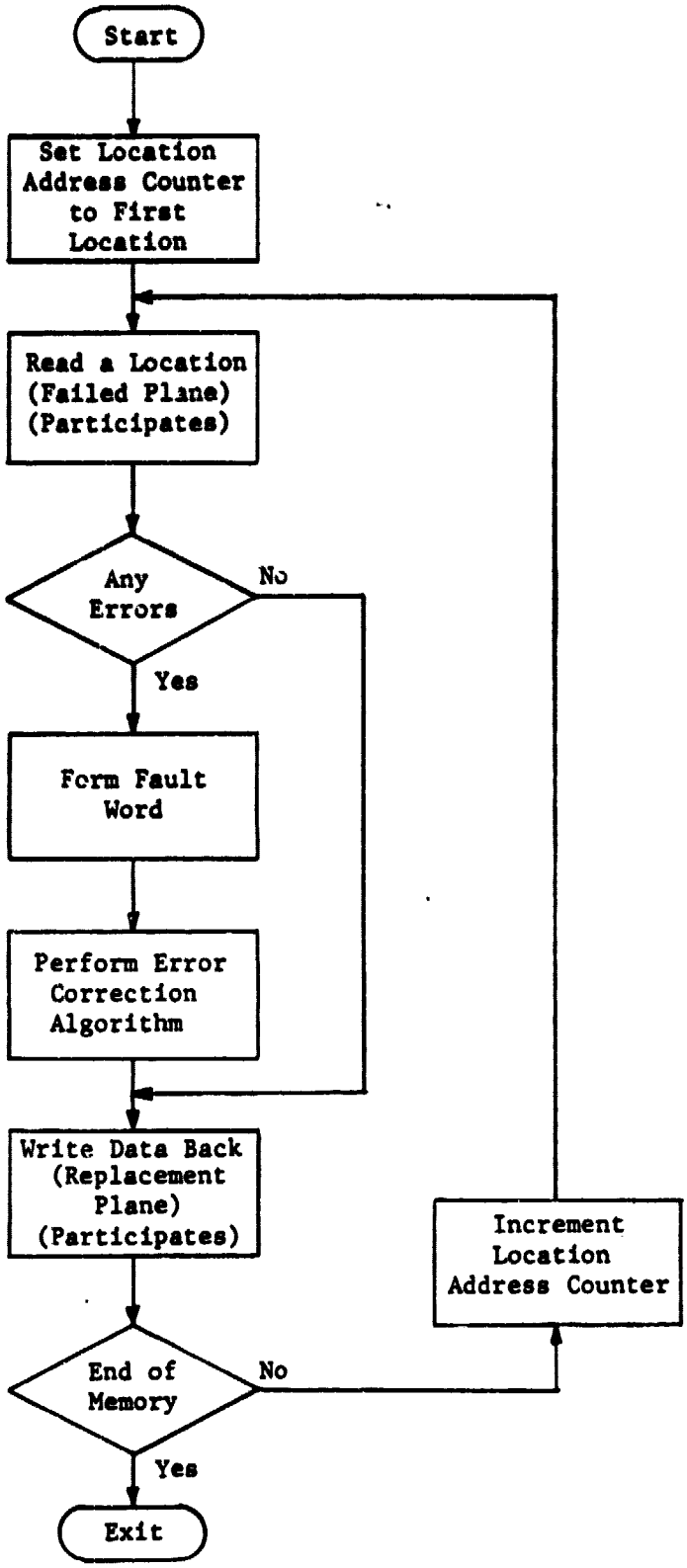


Figure 3-15. TEST/COPY/CORRECT ROUTINE

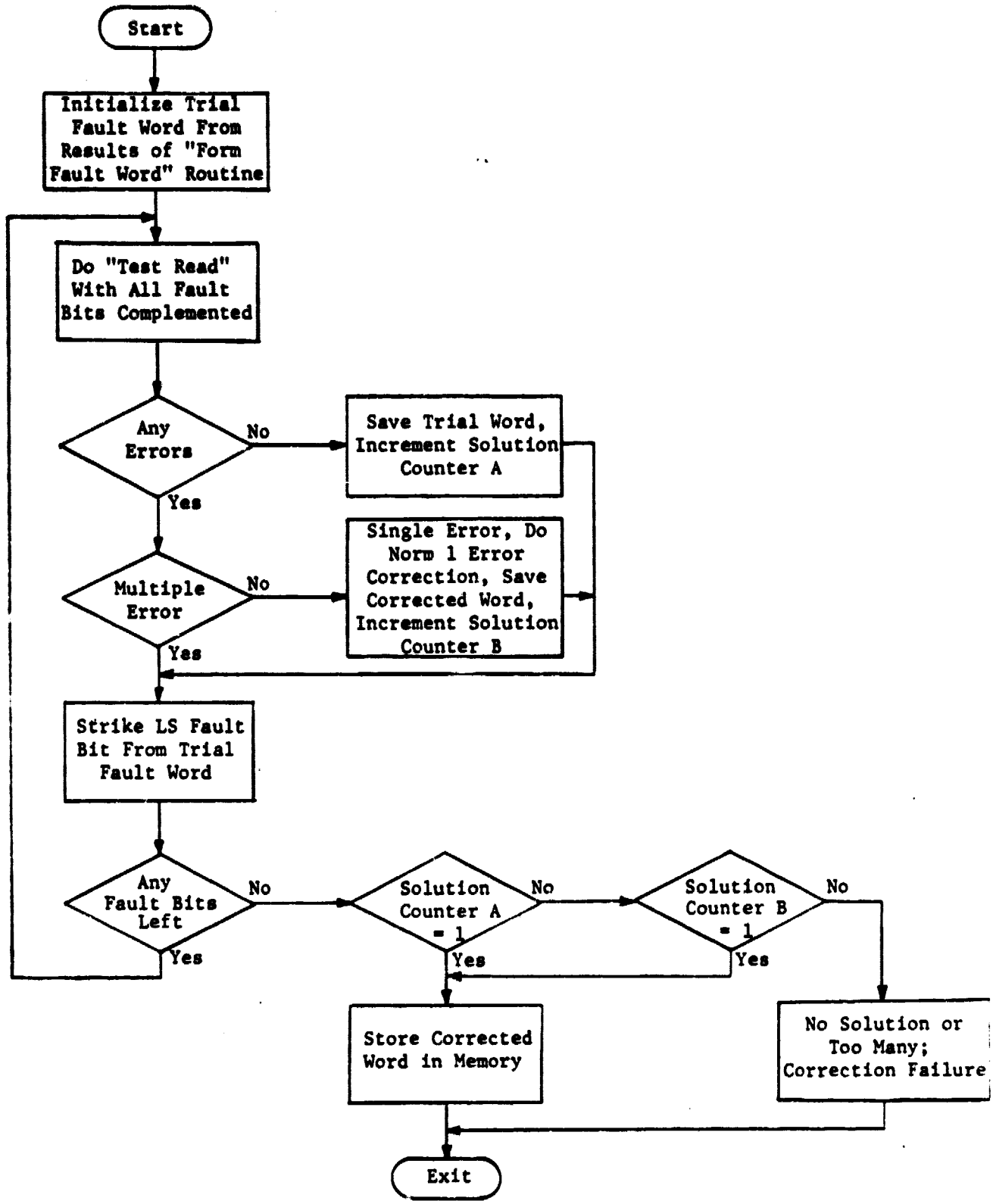


Figure 3-16. MULTIPLE ERROR CORRECTION ALGORITHM

usable life of the storage array, and (3) enhancing the overall reliability of the SUMC-IIC computer. Associated with this benefit is the recurring cost of time. Whenever a double error occurs, the microcode algorithms which locate the error, reconfigure memory and correct data errors, take the CPU "off-line" for the duration of time that it takes to execute the algorithms. Table 3-1 lists the execution time for each of the microcode routines previously discussed. As per the example presented in Table 3-1, 1 double faulted location in a 64K memory size machine would take the CPU "off-line" for 687 m sec. It is conceivable that at certain "mission critical" periods of time, 687 m sec of "off-time" would be intolerable. The risk of this occurring cannot be completely eliminated, but it can be greatly reduced by managing the tools which the FTM system provides.

The management concept is this: on a periodic basis during non-critical mission phases, enter via the DIAGNOSE instruction the Fault Tally and Address Fault Tally microcode routines. This provides the advantages of (1) the flight programmer selects the time periods for fault location and correction and (2) greatly minimizing the probability of a double error occurrence. To prevent the occurrence of the accumulation of "soft errors," the flight program should, on a periodic basic, read and rewrite all memory locations. This could be done in increments of 1K or 4K or 16K bytes. This will further minimize the probability of a double error occurrence. The disadvantage of this concept is the increased software overhead but, when weighed against a 687 m sec. "off-line" time during "mission critical" phases, it seems reasonable to conclude that the FTM management concept should be included as a part of the flight software.

Table 3-1

FTM DIAGNOSTIC EXECUTION

	DIAGNOSTIC	EXECUTION TIME
A.	Fault Tally	62 μsec + 19 μsec/(halfword location)
B.	Address Fault Tally	253 μsec/4K block
C.	Reconfigure	7 μsec
D.	Copy/Test/Correct	5 μsec/halfword location + 62 μsec/ halfword location if correction needed)
E.	Multiple Error Overhead	390 μsec

TOTAL EXECUTION TIME FOR 64K MACHINE WITH 1 DOUBLE FAULTED LOCATION

(A) 522654 μsec + (C) 7 μsec + (D) 164902 μsec + (E) 390 μsec = 687 m sec

3.3 IMPLEMENTATION OF ADDRESS EXTENSION

Expanding the ALU and appropriate data paths and registers was selected as the approach to address expansion for the SUMC-IIC. This provides the addition of 20 bit numbers when address calculations are being done as a part of the effective address calculation (EA Calc).

The EA Calculation is: E A = D + (B) + (X) where D is the 12 bit displacement from the instruction register, (B) is the contents of one of the general registers (now 20 bits) and (X) is the contents

of one of the general registers (20 bits). This addition is performed by calculating an interim number INTR = D + (B) then getting the final EA=INTR + (X).

Performing the 20-bit arithmetic for either INTR or EA requires simultaneous access to 20 bits of the SPM (where the general registers are implemented) and a 20 bit wide ALU for adding positive integers. Since the D is a positive integer, only the extended part of the ALU does not have to propagate any negative signs from the lower part of the ALU.

Figure 3-17 shows the parts of the SUMC-IIC block diagram which are affected by the expanded addressing. The salient features of this hardware are discussed below:

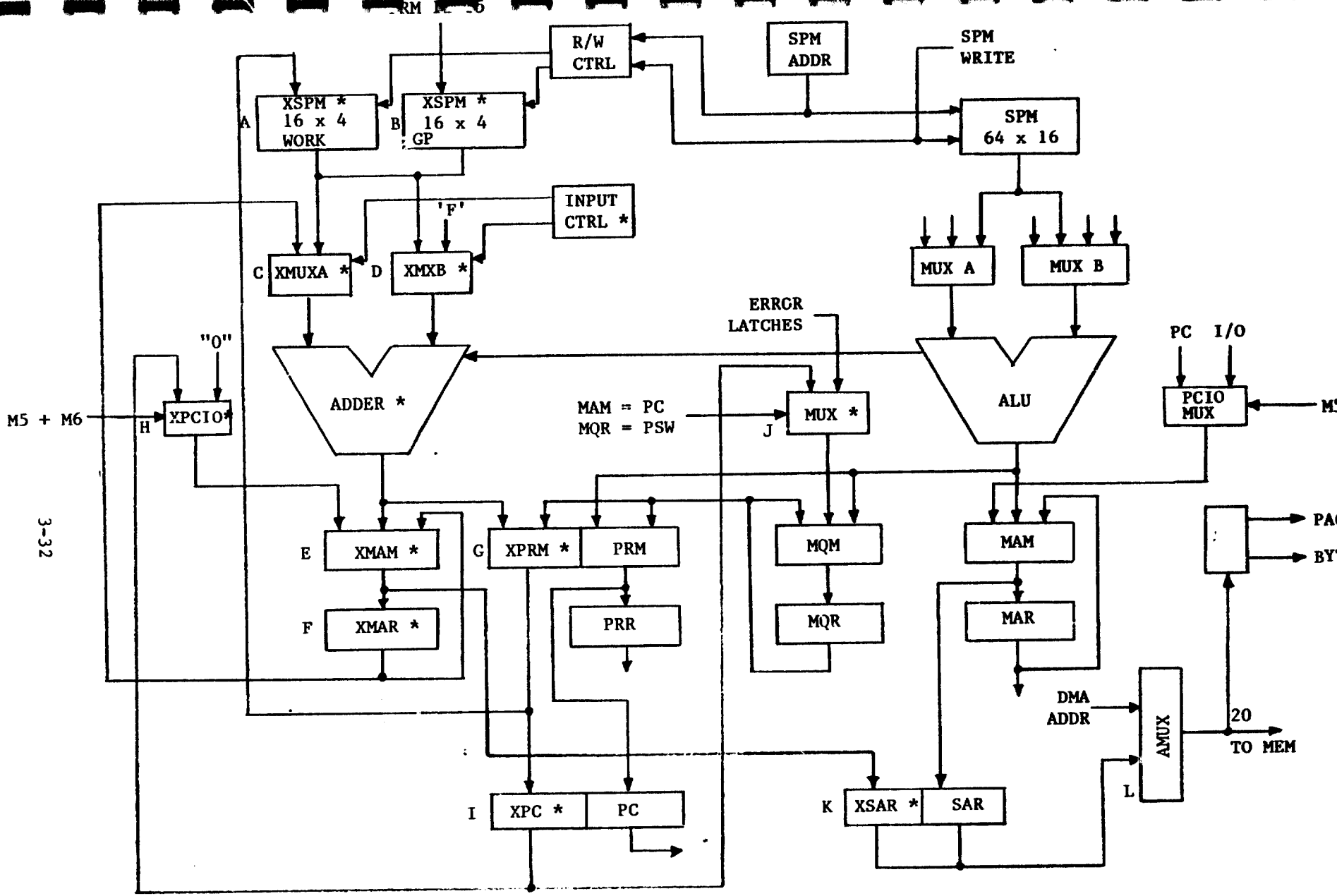
- o A four bit extension to the PC was added to accommodate the 20-bit addressing.

- o The extension to the PC can be loaded from the extended ALU for branch instructions the same as the regular part of the PC can be loaded from the PRM.

- o The PC extension also goes through a new four bit MUX into the SAR extended for instruction fetching.

- The PC extension can be read into the data path (MQM) through a new four bit MUX. This is used for BAL and store PSW type operations.

- o The address MUX was expanded to handle the 20 bit address both from the SAR and from DMA.



* NEW HARDWARE

Figure 3-17. EXTENDED ADDRESSING - DATA PATH EXTENSION

- o The address decode logic was expanded to provide six page select signals, for up to 96K bytes in the prime CPU box, and the full address is available to be sent to an external memory unit so that page selects can also be generated externally.

- o The storage protect registers (not shown) were expanded to 1024 X 2 to accommodate the 1024 segments of 1024 bytes each associated with the 20 bit address capability.

- o Manipulation of SPM for 20 bit arithmetic is not as straight forward as the items just mentioned and it is described in the immediately following paragraphs.

To form the intermediate sum INTR = D + (B) in the extended ALU, the most significant four bits (MSB) of the base address, B must be added to the zeros which represent the MSBs of the 12-bit D field. This poses a problem, however, since the 32 bit general register holding the 20 bit base value is located in two separate 16-bit locations in the SPM. Two SPM locations cannot be read at the same time so a portion of the SPM is duplicated. Therefore, when the SPM is reading the least significant 16 bits of a general register, the SPM extension must be reading what amounts to the least significant four bits of the most significant half of the general register. Thus the 20 bit base address (or index register) is made available simultaneously to the ALU and extended ALU. The ALU extension register, which is cleared, is fed back to the ALU extension to provide the most significant zeros corresponding to the upper part of D (which is only 12 bits). At the end of this cycle, the ALU extension register contains the four MSBs of the 20-bit Base Register plus any carry from the displacement

addition. The second addition is like the first except the interim sum (20 bits) is added to the 20-bit X register value. The full 20-bit EA is loaded into either the PC or the SAR according to the instruction being executed.

The next problem is getting the MSB portion of the general registers into the SPM extension. Since the general registers are loaded in a double precision manner (first the lower half then the upper half) the extended SPM must be loaded when the upper half of the register is loaded. Thus special manipulation of the most significant bits of the SPM address is used to load when an upper half general register is being loaded and read at all other times.

The overall operation has been described above. Table 3-1 contains the control equations for the block shown in Figure 3-17. For definition of some of the signals make reference to the microprogram control word. For example: S 1 is the first bit of the SPM field, and R 2 is the second bit of the REGISTER field.

Table 3-2. CONTROL EQUATIONS

Block	Control
A	Read $\overline{S1} \cdot \overline{S2}$
	Write $\overline{S1} \cdot \overline{S2} \cdot$ SPM write
B	Read S1 = 1
	Write S1 $\cdot \overline{S2} \cdot$ SPM Write
C	Output = MAR if MUXA = MAR (A1 $\cdot \overline{A2} \cdot$ A3)
	Output = SPM if MUXA = SPM (A1 $\cdot \overline{A2} \cdot \overline{A3}$)
	Else Output = 0
D	Output = SPM if MUXB = SPM (A5 $\cdot$ A6 $\cdot \overline{A7}$)
	Output = F if ALU = A-B ($\overline{A8} \cdot$ A9 $\cdot \overline{A10}$)
	Else Output = 0
E	Output Enable MAM = MAR or MAM = ALU or MAM = PC ($\overline{R3} \cdot \overline{R4}$)
	Output = XMAR if MAM = MAR ($\overline{R1} \cdot$ R2 $\cdot \overline{R3} \cdot \overline{R4}$)
	XALU if MAM = ALU (R1 $\cdot \overline{R2} \cdot \overline{R3} \cdot \overline{R4}$)
	XPC if MAX = PC (R1 $\cdot$ R2 $\cdot \overline{R3} \cdot \overline{R4}$)
F	Load = R5 $\cdot$ CKZ (No reset)
G	Output = MQR Bits 12-15 if MQR to PC EXT ($\overline{M7} \cdot$ M8 $\cdot \overline{M9} \cdot$ M10)
H	Output = 0 if MAM = PC10 (M5 + M6 = 1)
	Else = PC EXT
I	Load = Load PC
J	Output = PC EXT if MQR = PSW (R11 $\cdot$ R12 $\cdot \overline{R13}$) and
	MAM = PC10 (M5 + M6 = 1)
K	Load = CPSAR Load Clock
	Reset = POR
L	Same as A MUX